GRAPHICAL SIMULATION OF CUTTER PATH USING SYMBOLIC CONJUGATE GEOMETRY

by

Neti Sasidhar

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

MARCH, 1995

GRAPHICAL SIMULATION OF CUTTER PATH USING SYMBOLIC CONJUGATE GEOMETRY

A Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

MASTER OF TECHNOLOGY

by

Neti Sasidhar

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

March, 1995

. . . to
my parents . . .

Certificate

This is to certify that the work contained in the thesis titled **GRAPHICAL SIMULATION OF CUTTER PATH USING SYMBOLIC CONJUGATE GEOMETRY** by **Neti Sasidhar (Roll No: 9311118)**, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

15/3/95

Prof. S.G.Dhande
Professor,
Department of Computer Science and Engineering
IIT, Kanpur

Acknowledgements

My first and foremost gratitude goes towards my guide Prof. S.G.Dhande, who took active interest in my thesis and whose course introduced me to the fascinating field of Computer Graphics and induced me to do my thesis in it.

Next, following the tradition, I would like to thank all my friends of all assorted shapes and sizes, wise and otherwise, for providing their company and making my stay at I.I.T - K more or less bearable, as the case might be.

As a fresh member of the (now famous) GAALI CLUB, I thank all its members for the wonderful pseudo-fundu discussions (they are out of scope of this work) I had with them. My special thanks, however, are reserved for the veterans and GURUS of this club, namely TAN, A.P.Avan, TRK and Shanks, who today, on the basis of sheer merit, occupy the highest posts in the club.

Many others, notably Mr. B.A.Bukhari and Mr. B.Rajak shall always occupy significant chunks of my memory.

I thank all those too, who, just by being, made me realize that it takes all sorts to make this world. At this historical moment, I thank Mr. Leslie Lamport for developing LATEX, because of which I discovered some little known keys of the keyboard.

-Sasy

Abstract

There has been an extensive research going on in the field of graphical simulation of manufacturing processes. Most often such simulations rely almost entirely on numerical methods. There is another, a symbolic, approach in which the values to variable entities are bound in the final steps, and the intermediate calculations are done in a symbolic fashion.

In this work such a simulation of machine tooling process is done. For this purpose a symbolic manipulator has been designed and developed which manipulates the constituent equations in a symbolic manner, and then the output, which is the mathematical counterpart of the actual surface, is rendered on a graphics workstation. Several examples have been taken and directly tested and the results were satisfactory.

The user is provided with a textual menu which prompts him to enter specifications for a cutter, the kinematics of motion of the blank and the cutter and some rendering options. The program then produces the output which he can see on the terminal. This package allows a designer to view a machining process before it is actually done. This enables him to avoid pitfalls, if any, and take corrective actions. This also saves a lot of time, effort and money which are invariably associated with the actual machining.

The present work finds applications in tool and cutter grinding, gear cutting, turbine blade cutting and many other similar machine tooling processes.

Contents

1 Introduction **4**

1.1 Geometric Modeling 4

1.1.1 Conjugate Surfaces 5

1.2 Literature Review 5

1.3 Objectives, Scope and Organization 7

2 Mathematical formulation **9**

2.1 Envelope Theory - 2D 9

2.1.1 Symbolic Model 12

2.1.2 Basic Concepts 12

2.2 The Generic Cutter 13

2.2.1 Introduction 13

2.2.2 Description of the generic cutter 14

2.2.3 The Algorithm 17

2.3 The Machined Surface 20

2.3.1 The Algorithm 27

2.4 Computational Model 31

2.5 Some Observations 32

3 Symbolic Manipulator **35**

3.1 **Role of symbolic manipulator** 35

3.1.1 Symbolic Manipulation – What is it? 35

3.1.2 History and development 35

3.1.3 Why a symbolic manipulator? 36

3.2 Software Design and Implementation 37

3.3 Some Pitfalls 44

3.4 Some Examples 45

4 Graphical Simulation **48**

4.1 Surface Rendering 48

4.1.1 Calculation of Normals 49

4.2 Animation 49

4.3 User Interface 50

4.4 Examples 51

5 Conclusions **58**

5.1 Considerations for Further Work 58

5.2 Some Techniques and Improvements 59

Bibliography **61**

A Formal Specification of Symbolic Manipulator **64**

B A Sample Input to Symbolic Manipulator **70**

List of Symbols

$\sum_1$ - Surface of cutter.

$\sum_2$ - Surface of blank.

S_0 - Fixed global coordinate system.

S_1 - Coordinate system attached to the cutter.

S_2 - Coordinate system attached to the blank.

a_1 - Distance of O_1 from O_0 along X_0 axis.

b_1 - Distance of O_1 from O_0 along Y_0 axis.

c_1 - Distance of O_1 from O_0 along Z_0 axis.

θ_1 - Rotation of coordinate system S_1 about X_0 axis.

ϕ_1 - Rotation of coordinate system about Y_0 axis.

ψ_1 - Rotation of coordinate system about Z_0 axis.

a_2 - Distance of O_2 from O_0 along X_0 axis.

b_2 - Distance of O_2 from O_0 along Y_0 axis.

c_2 - Distance of O_2 from O_0 along Z_0 axis.

θ_2 - Rotation of coordinate system S_2 about X_0 axis.

ϕ_2 - Rotation of coordinate system S_2 about Y_0 axis.

ψ_2 - Rotation of coordinate system S_2 about Z_0 axis.

u , v - Parameters describing the surface $\sum_1$.

${}^1\vec{p}$ - Point P with reference to the tool coordinate system S_1.

${}^2\vec{p}$ - Point P with reference to the blank coordinate system S_2.

${}^0\vec{n}_{P,1}$ - Unit normal vector of the cutter surface $\sum_1$ at point P with reference to coordinate system S_0.

${}^{1}\vec{n}_{P,1}$ - Unit normal vector of the cutter surface $\sum_1$ at point P with reference to coordinate system S_1.

${}^{0}\vec{n}_{P,2}$ - Unit normal vector of the blank surface $\sum_2$ at point P with reference to coordinate system S_0.

${}^{2}\vec{n}_{P,2}$ - Unit normal vector of the blank surface $\sum_2$ at point P with reference to coordinate system S_2.

$[{}^{0}_{1}M]$ - Homogeneous transformation matrix of a vector from the coordinate system S_1 to S_0.

$[{}^{0}_{2}M]$ - Homogeneous transformation matrix of a vector from the coordinate system S_2 to S_0.

${}^{0}\vec{V}_{P,1}$ - Velocity of the surface of the cutter with reference to the coordinate system S_0.

${}^{0}\vec{V}_{P,2}$ - Velocity of the surface of the blank with reference to the coordinate system S_0.

${}^{0}\vec{V}_{P,12}$ - Relative velocity vector between the tool and blank with reference to coordinate system S_0.

List of Figures

2.1 Envelope of a Family of Curves. 10
2.2 Definition of a Generic Cutter 15
2.3 Ball Nose End Mill Cutter 21
2.4 90 deg. Tool 22
2.5 Side and Bottom Angle Cutter 23
2.6 Non-Fillet Type End Mill 24
2.7 Conical Cutter 25

3.1 The Symbolic Manipulator 38

4.1 The Conical Cutter 54
4.2 The Helical Screw being Machined 54
4.3 The Helical Screw 55
4.4 The Ball Nosed End Mill Cutter 55
4.5 The Spiral Screw being Machined 56
4.6 The Spiral Screw Surface 56
4.7 The Machining of Star Shaped Surface 57
4.8 The Star Shaped Surface 57

List of Tables

4.1 The Cutter Parameters for the Conical Cutter 51

4.2 The Cutter Kinematics of the Conical Cutter 53

4.3 The Blank Kinematics for the Conical Cutter 53

4.4 The Cutter Kinematics of the End Milling Cutter 53

4.5 The Blank Kinematics for the End Milling Cutter 53

4.6 The Cutter Parameters for Bottom Angle Cutter 53

4.7 The Cutter Kinematics of the Bottom Angle Cutter 53

4.8 The Blank Kinematics for the Bottom Angle Cutter 53

Chapter 1

Introduction

1.1 Geometric Modeling

The modeling of the geometric and topological features comes under the field of geometric modeling. Geometric information is essential for describing products, because every machine functionality is realized one way or the other by its geometric features.

There are basically three levels of geometric modeling based on the dimensionality of the representational domain. These levels are :

- Wireframe modeling.
- Surface modeling.
- Solid modeling.

Out of these three, solid modeling has the most explicit representation of the objects in the real world and implicitly embodies the lower levels of representation as well. To make the concept clearer, we give a simple example.

We know that the equation

$$x^2 + y^2 + z^2 \leq r^2$$

explicitly represents a solid sphere centered at origin and with radius r. It is easy to see that the same equation implicitly contains the equation of the surface of the same sphere.(Just change the $\leq$ to $=$.) This was a very primitive example. In general it is not necessarily true that any solid modeling approach can uniformly manipulate

entities of lower dimensionality as well as solids. Formerly there was a notion that for product shape description solid modeling framework is the best. However, it has been realized that solid modeling is not the panacea for handling product shapes. For example, wireframe models are useful for analysis of mechanisms whereas surface modeling is appropriate for designing of machining applications. In fact, the present work relies on surface modeling extensively.

1.1.1 Conjugate Surfaces

In many situations, we encounter surfaces which are not independent but related to one another by some constraint. This is more so in the case of machining processes, where one surface is constantly moulding the other. In the present situation, the surfaces $\sum_1$ and $\sum_2$ are related by what is known as the *higher pair contact* constraint also known as *conjugate surface* constraint. These are the surfaces which have a common normal at the point of contact.

In cutter path simulation the two surfaces $\sum_1$ and $\sum_2$ are conjugate to one another because the machined surface is nothing but different instances of the cutter profile. This will become clearer in subsequent chapters. In practice, however, there will be slight deviations because of the non-ideal nature of the cutter and blank materials.

1.2 Literature Review

One of the major areas of research in manufacturing is the simulation of machining processes. The two basic determinants of a machining process are the cutter geometry and the geometry of the relative motion between the cutter and the work piece. There are many kinematic chain models of relative motion in literature. Srinivasan and Ferreira (1990) have proposed a two level model for machining processes. The lower level models the kinematic chain and the higher level decomposes the relative motion into cutting and feed motion components and with each component a subset of kinematic chain from lower level is associated. They have shown that for a significant subset of all machining processes their approach is useful in feature definition and validation, process selection and kinematic planning at the machining center.

There has been a lot of development in the area of geometric modeling, particularly solid modeling which has provided design and manufacturing engineers with elegant and precise definition of components or products that are to be designed (Mortenson 1985; Mantyla 1988; Voelcker and Requicha, 1977; Faux and Pratt, 1988). Based on these definitions, several research workers have developed methodologies to generate process plans for manufacturing the designed components (Chang and Wysle, 1985; Kanumury and Chang, 1991; Joshi and Chang, 1988; Perng et. al,1990; Shah, 1991; Srinivasan and Ferreira,1990). Most of the process plans are based on the geometric features of a component and the capability of the manufacturing process to produce those features. A lot of research has been done regarding the physics and mechanics of several processes (Acherkan et. al., 1973; Ghosh and Mallik, 1981; Ber et.al., 1988; Sen and Bhattacharya, 1969; Pande and Shah, 1980) but most of it is directed towards the area of gear manufacturing.

Wang and Wang (1986 a, 1986 b) have defined swept volume using envelope theory approach. However they don't model the kinematic structure of the machine tool explicitly. Yap (1988) has given some algorithms for computing 3-D tool paths for planar and non-planar machined surfaces. Dhande et al. (1990, 1992) have shown how the geometry of a machined surface can be derived from the wire-EDM, which is a non-traditional manufacturing process. Drysdale and Jerard (1989) have done a discrete simulation of machining for sculptured surfaces. Blackmore and Len (1990) have proposed a swept differential equation approach to define the swept volume. Generalized sweep operations are discussed by K.Weiler and D.Melachlan (1990).

The problem of conjugate geometry between a pair of moving as well as contacting surfaces is termed as higher pair contact problem in kinematics (Boltyanskii, 1964; Dudley and Darle, 1962; Chakraborthy and Dhande, 1977). These problems have been analyzed for design of cams and gears. Dhande and Chakraborthy, 1977, have given methods to define kinematics of relative motion as well as the geometry of contacting surfaces for 3-D cams. Some work has also been done concerning the conjugate geometry of planar and spatial gear mechanisms (Dudley, 1962; Dhande and Chakraborthy, 1977).

One of the major computational difficulties in formulating and analyzing the problems of conjugate geometry is that for every given situation of a pair of contacting surfaces and their relative motion, one has to derive the biparametric equations of the surfaces every time. This can be improved by using the symbolic manipulation approach for the problems of conjugate geometry , especially for modeling different manufacturing processes. Reinholtz and Dhande et al.(1990) have described an approach of how symbolic manipulation can be used for defining the machined surfaces. They have considered only a three degrees of freedom model to show how helically swept surfaces can be manufactured using alternative strategies.

1.3 Objectives, Scope and Organization

The objective of the present work is to graphically render the path taken by the cutter, given the parameters describing its surface and kinematics regarding the relative motion of the cutter and the blank with all the intermediate computations done in symbolic form.

This report consists of five chapters. Chapter 1 briefly introduces the field of geometric modeling and conjugate geometry followed by an extensive literature review.

Chapter 2 explains the problem, drawing analogy from 2-D envelope theory. Then the symbolic model of the generic cutter and the symbolic algorithm to derive the cutter path (in other words, the machined surface) are presented. And finally the computational part of the algorithm is explained followed by some observations on the limitations of this (envelope theory) approach.

Chapter 3 starts with the role of the symbolic computation in the present work and goes into the design and implementation aspects of the symbolic manipulator specifically designed for this work. At the end some examples are given to illustrate the capabilities of the symbolic manipulator.

Chapter 4 is about the rendition of surfaces on a graphics workstation. It also discusses the interface provided to the user and some tips on how to use it. It concludes with with some examples which have been tested and run on a graphics workstation.

Chapter 5 concludes the work with some technical details and suggestions for further extension of this work.

Chapter 2

Mathematical formulation

2.1 Envelope Theory - 2D

In this section we will give a brief summary of the important results of the envelope theory.

A family of curves can be specified by the implicit form equation

$$f(x,\ y,\ \alpha)\ =\ 0 \tag{2.1}$$

where x and y are the Cartesian coordinates of a generic point and α is a parameter (See Fig.2.1). By giving a specific numerical value of α, say α_i, one can obtain the equation of a member of the family as

$$f(x,\ y,\ \alpha_i)\ =\ 0 \tag{2.2}$$

It is assumed that every member of the family is a C^1-continuous (A C^1 -continuous curve is one which has first differential at all points on it) curve. Furthermore, it is assumed that successive members of the family intersect at a finite point. Each of these points of intersections, it is assumed, has a limiting position when the successive member curves are considered infinitesimally close and approaching a limit (Boltyanskii, 1964). The point of intersection of a pair of infinitesimally close member curves can be obtained by solving the equations

$$f(x,\ y,\ \alpha_i)\ =\ 0 \tag{2.3}$$

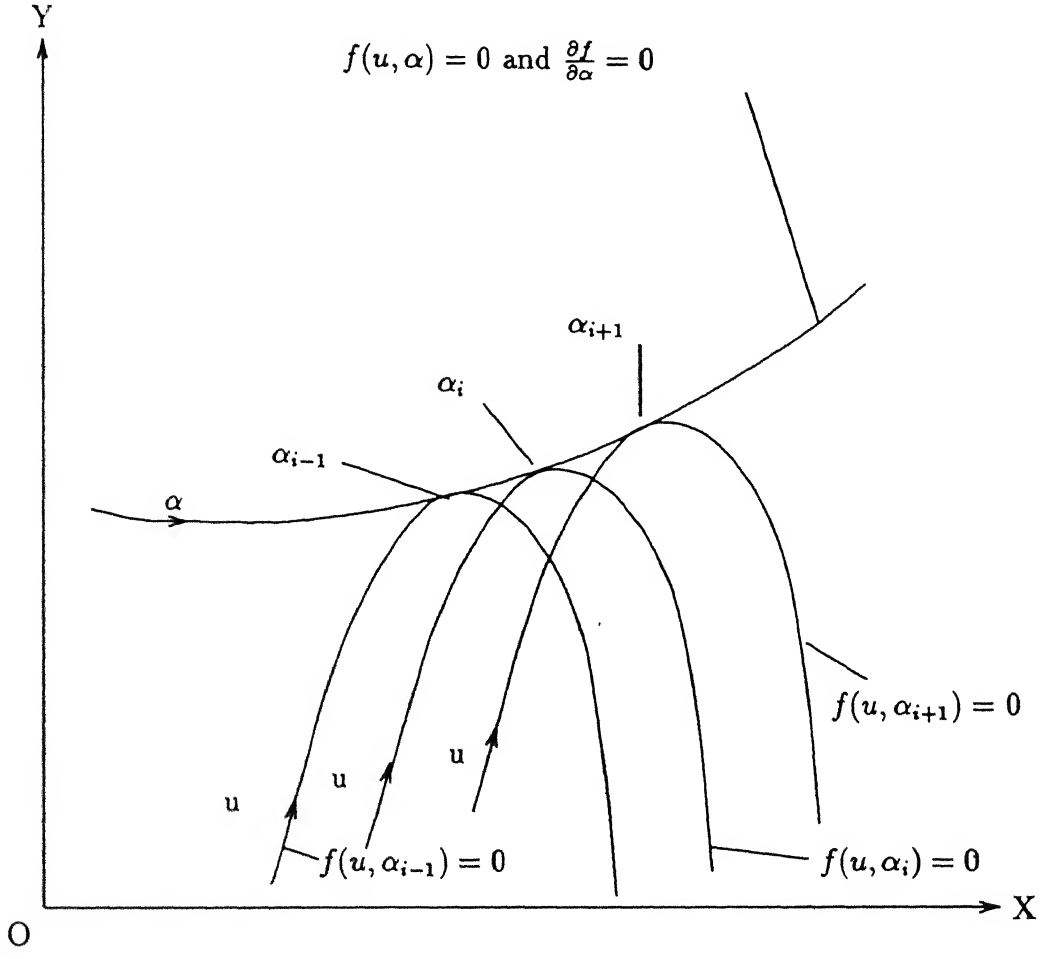


Figure 2.1: Envelope of a Family of Curves.

$$f(x,\ y,\ \alpha_i + \Delta\alpha)\ =\ 0 \tag{2.4}$$

$\Delta\alpha$ being an infinitesimal increase in α. The limiting position of this point of intersection can be obtained by eliminating α from the following equations:

$$f(x,\ y,\ \alpha)\ =\ 0 \tag{2.5}$$

$$\frac{\partial f}{\partial \alpha}\ =\ 0 \tag{2.6}$$

All such limiting positions, in turn, define the envelope to the family of curves. In parametric form, this equation of a family of curves is given by

$$f(u,\ \alpha)\ =\ 0 \tag{2.7}$$

and a generic point P of this can be expressed as

$$\vec{p}(u,\ \alpha)\ =\ [x(u,\ \alpha)\ \ y(u,\ \alpha)] \tag{2.8}$$

where $u_{min} \leq u \leq u_{max}$ *and* $\alpha_{min} \leq \alpha \leq \alpha_{max}$. In parametric form, the limiting points of intersections can be obtained by considering the collinearity of the vectors

$$\frac{\partial \vec{p}}{\partial u}\ and\ \frac{\partial \vec{p}}{\partial \alpha}.$$

In other words, the equation of the envelope in parametric form can be obtained by eliminating the parameter u between the following two equations (Chakraborty and Dhande, 1977):

$$f(u,\ \alpha)\ =\ 0 \tag{2.9}$$

$$\vec{n} \cdot \vec{V}\ =\ 0 \tag{2.10}$$

where

$$\vec{V} = \frac{\partial \vec{p}}{\partial \alpha}\ and\ \vec{n}\ \cdot\ (\frac{\partial \vec{p}}{\partial u})\ =\ -1\ .$$

It will be in place to give a physical interpretation to Equation (2.9). If the parameter α represents time then $\vec{V}$ and $\vec{n}$ will be the velocity and normal vectors at any point. When a curve is being swept, every point on the curve is moving along the direction of its velocity. If this velocity vector is orthogonal to the normal vector of the curve at any point (the C^1-continuity ensures that there would be atleast one such point) then that point of the curve is the point defining the envelope curve at that instant. All other points will lie within the swept area of the curve.

The above description was for the envelope of a curve in a fixed frame of reference. Many times it is required to find the envelope of a curve or a surface in another frame of reference which is moving with respect to the frame of reference in which the curve is moving. In chapter 3 we talk about one such case.

2.1.1 Symbolic Model

In any machining process there are three elements which describe the machining process completely (assuming ideal behavior of cutter and blank material). They are :

1. The surface of the cutting tool (denoted by $\sum_1$ throughout this work).
2. The surface to be produced, denoted by $\sum_2$ and
3. The relative motion between the above mentioned surfaces.

To be able to produce the the surface $\sum_2$ properly, it is necessary to assume that the two surfaces $\sum_1$ and $\sum_2$ are conjugate to each other. Or in other words, these two surfaces maintain a higher pair contact throughout the cutting motion. In the following sections we will give the symbolic models for the generic cutter ($\sum_1$) as well as the developed surface ($\sum_2$). But let us first recapitulate the symbols used in the derivations.

2.1.2 Basic Concepts

A generic model of a machine tool is essentially a representation of the kinematic structure of the machine tool (Acherkan, 1973). The kinematic structure of a machine

tool describes the kinematics needed for giving the cutter its cutting motion, and the feed motions to the cutter and/or the blank.

In the present model (Karunakara Poopathi, K.P, 1993), the cutter and the blank are considered to be free bodies in space and therefore their positions and orientations (w.r.t the coordinate axes of a fixed frame of reference) can be described by twelve functions (of time), six for each surface. The fixed frame of reference is denoted by S_0 (O_0 – X_0, Y_0, Z_0). A coordinate system S_1 (O_1 – X_1, Y_1, Z_1), is assumed to be attached to the cutter, and another coordinate system S_2 (O_2 – X_2, Y_2, Z_2), to the blank. The translation of the origins O_1 and O_2 of the moving coordinate frames S_1 and S_2 respectively are (a_1, b_1, c_1) and (a_2, b_2, c_2) whereas $(\theta_1, \phi_1, \psi_1)$ and $(\theta_2, \phi_2, \psi_2)$ denote the rotation of the reference frames around the three axes of the fixed coordinate frame.

2.2 The Generic Cutter

2.2.1 Introduction

The geometry of a generic cutter can be modeled as an axisymmetric, biparametric surface. The axis of symmetry depends on whether the cutter is end mill type cutter, a side mill type cutter, a face mill type cutter or any combination of these basic types of cutter.

In numerical control literature, a generic cutter is defined using seven parameters and the generatrix consists of two straight line segments forming the tip flank as well as the side flank and a circular fillet between these straight line segments (Chang, 1989). In this work, an extension of this is used (Karunakara Poopathi, K.P, 1993). The generatrix curve is assumed to be a piecewise continuous curve consisting of three line segments and a circular fillet between the tip flank and the side flank lines. This generatrix curve can be defined by means of eight parameters, viz.,*d, r, e, f, a, b, h* and *h1*. The cutter surface can be obtained by rotating the generatrix curve either along Z_1 axis or along the X_1 axis. Rotation about Z_1 axis produces various types of end mill cutters whereas rotation around X_1 axis generates a disc type cutting

tool geometry which will represent a side-and-face milling cutter or a grinding wheel. If the directrix is helical, then the geometry of a gear hob, or a tapping tool can be obtained. Thus by choosing appropriate values of the parameters describing the planar curve and the sweep motion, one can obtain a variety of conventional and non-traditional cutting tools.

2.2.2 Description of the generic cutter

The planar curve of the generic cutter surface consists of three straight line segments and a circular arc segment. For the case of end mill type cutter, the first and second line segments will generate the tip and side flanks of the cutter while the third straight line segment will generally represent the shank of the cutter. The circular segment will generate the corner radius of the end mill. For this purpose, a coordinate system S_c (O_c – X_c, Y_c, Z_c) is considered to be attached to this cutter profile. This planar profile of the cutter can be represented by means of a parameter u which is the profile length of any point on the profile with respect to origin O_c. The cutter surface $\sum_1$ can be obtained by sweeping the this generatrix profile along a directrix curve which can be circular, helical etc. This sweep motion is represented by by means of a homogeneous transformation matrix $[\,{}^1_cM]$ whose elements will be functions of parameter v.

$$O_cP : z_c = x_c \tan a \tag{2.11}$$

$$PQ : (x_c - e)^2 + (z_c - f) = r^2 \tag{2.12}$$

$$QS : x_c = z_c \tan b - \frac{d}{2}\tan a \tan b + \frac{d}{2} \tag{2.13}$$

$$ST : x_c = h \tan b - \frac{d}{2}\tan a \tan b + \frac{d}{2} \tag{2.14}$$

where

O_cP - First straight line segment,

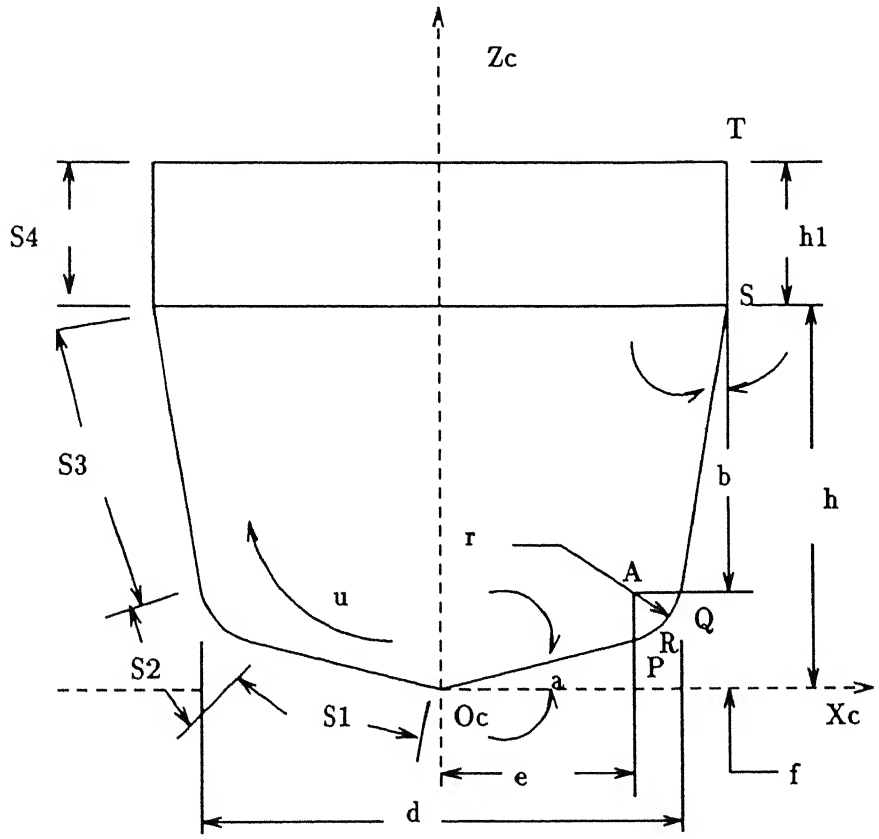


Figure 2.2: Definition of a Generic Cutter

PQ - The circular segment with center at A,

QS - Second straight line segment,

ST - Third straight line segment,

R - Point of intersection of the segment O_cP and the third segment QS,

d - Diameter of the cutter measured at R,

r - Radius of the circular segment. This is positive for a convex arc and negative for a concave arc,

a - Angle of segment O_cP measured from X_c axis.

$$\frac{-\pi}{2} < a < \frac{\pi}{2},$$

b - Angle of segment QS measured from Z_c axis.

$$\frac{-\pi}{2} < b < \frac{\pi}{2},$$

e - X coordinate of point A in the coordinate system S_c,

f - Z coordinate of point A in the coordinate system S_c,

h - Nonparallel height of the cutter,

h_1 - Parallel height of the cutter,

is_fillet_radius - Flag which is to be set to $TRUE$ when the circular segment PQ is tangent to both the straight line segments O_cP and QS and $FALSE$ otherwise,

u - Parameter which is the profile length of any generic point P on the profile measured from O_c,

v - Parameter which describes the sweep of the two dimensional profile,

$[{}^1_cM]$ - Homogeneous transformation matrix which describes the sweep of the two dimensional profile,

s_1 - Length of the first straight line segment O_cP,

s_2 - Length of the circular arc segment PQ,

s_3 - Length of the second straight line segment QS,

s_4 - Length of the third straight line segment ST,

${}^c\vec{p}$ - Position vector of a generic point P on the two dimensional profile of the cutter in S_c coordinate system in terms of u and

${}^1\vec{p}$ - Position vector of a generic point P on the surface of the cutter in S_1 coordinate system in terms of u and v.

The parametric equation of the generatrix curve ${}^c\vec{p}(u)$ with respect to the coordinate system $S_c(O_c - X_c, Y_c, Z_c)$ can now be defined as :

$$ {}^1\vec{p}(u,v) = {}^c\vec{p}(u) \cdot [\,{}^1_cM] \tag{2.15} $$

where $[\,{}^1_cM]$ is the matrix describing the sweep motion.

2.2.3 The Algorithm

In this section we will give the algorithm used to derive the biparametric equation of the surface $\sum_1$ from the nine parameters of the cutter profile described earlier (Karunakara Poopathi,K.P, 1993). It has been implemented in the symbolic manipulator designed in this work. It may be noted that all the parameters are not needed in all cases. The algorithm may be stated as follows :

1. Calculate e and f if $is\_fillet\_radius = TRUE$.

$$ e = \frac{d}{2} - r\left(\frac{\cos a - \sin b}{\cos(a+b)}\right) \tag{2.16} $$

$$ f = \frac{r + e\sin a}{\cos a} \tag{2.17} $$

2. Calculate point P.

$$ x_c^P = (e + f\tan a)\cos^2 a - $$

$$ \sqrt{[(e + f\tan a)\cos^2 a]^2 - (e^2 + f^2 - r^2\cos^2 a} \tag{2.18} $$

$$ z_c^P = x_c^P \tan a \tag{2.19} $$

3. Calculate point Q.

$$Z_c^Q = \{f + [e + \frac{d}{2}(\tan a \tan b - 1)] \tan b\} \cos^2 b + \sqrt{[\{f + [e + \frac{d}{2}(\tan a \tan b - 1)] \tan b\} \cos^2 b] - \{f^2 + [e + \frac{d}{2}(\tan a \tan b - 1)]^2 - r^2\} \cos^2 b} \tag{2.20}$$

$$x_c^Q = z_c^Q \tan b - \frac{d}{2}(\tan a \tan b - 1) \tag{2.21}$$

4. Calculate h if $is\_fillet\_radius = TRUE$ and $b = 0$.

$$h = z_c^Q \tag{2.22}$$

5. Calculate point S.

$$x_c^S = h \tan b - \frac{d}{2} \tan a \tan b + \frac{d}{2} \tag{2.23}$$

$$z_c^S = h \tag{2.24}$$

6. Calculate s_1, s_2, s_3 and s_4.

$$s_1 = \sqrt{(x_c^P)^2 + (z_c^P)^2} \tag{2.25}$$

$$s_2 = r \left\{ \arctan\left(\frac{z_c^Q - f}{x_c^Q - e}\right) - \arctan\left(\frac{z_c^P - f}{x_c^P - e}\right) \right\} \tag{2.26}$$

$$s_3 = \sqrt{(x_c^S - x_c^Q)^2 + (z_c^S - z_c^Q)^2} \tag{2.27}$$

$$s_4 = h_1 \tag{2.28}$$

7. Define the parametric equation

$${}^c\vec{p}(u) = [x_c(u)\ y_c(u)\ z_c(u)\ 1]$$

where,

for the interval $0 \leq u \leq s_1$

$$x_c(u) = u \cos a$$

$$y_c(u) = 0 \tag{2.29}$$

$$z_c(u) = u \sin a$$

for the interval $s_1 \leq u < s_1 + s_2$,

$$x_c(u) = e + abs(r) \cos\left\{\arctan\left(\frac{s_1 \sin a - f}{s_1 \cos a - e}\right) + \left(\frac{u - s_1}{r}\right)\right\}$$

$$y_c(u) = 0 \tag{2.30}$$

$$z_c(u) = f + abs(r) \sin\left\{\arctan\left(\frac{s_1 \sin a - f}{s_1 \cos a - e}\right) + \left(\frac{u - s_1}{r}\right)\right\}$$

for the interval $s_1 + s_2 \leq u < s_1 + s_2 + s_3$,

$$x_c(u) = [u - (s_1 + s_2 + s_3)] \sin b + x_c^S$$

$$y_c(u) = 0 \tag{2.31}$$

$$z_c(u) = [u - (s_1 + s_2 + s_3)] \cos b + z_c^S$$

and for the interval $s_1 + s_2 + s_3 \leq u \leq s_1 + s_2 + s_3 + s_4$,

$$x_c(u) = x_c^S$$

$$y_c(u) = 0 \tag{2.32}$$

$$z_c(u) = [u - (s_1 + s_2 + s_3)] + z_c^S.$$

8. Define the sweep matrix $[{}^1_cM]$

9. Define ${}^1\vec{p}(u, v)$

$${}^1\vec{p}(u, v) = {}^c\vec{p}(u) \cdot [{}^1_cM] \tag{2.33}$$

We give some examples of descriptions of the generic cutter. The input parameters, the symbolic output of the symbolic manipulator and the corresponding profile of the cutter are shown. The data are manipulated so that they are in more readable form.

2.3 The Machined Surface

In the earlier sections we saw how a generic cutter is described and how the various motion parameters are represented in the symbolic model. In this section we will give the necessary equations to synthesize the symbolic equations for the machined surface $\sum_2$.

As the equations of the generic cutter and the kinematics are known, it is possible[1] to derive the symbolic equation(s) for the path of the cutter, which is nothing but the required machined surface $\sum_2$. The cutter surface is expressed as a biparametric surface in u and v. As the two surfaces are assumed to be ideal and in higher pair contact, at the point of contact P we have,

$$ {}^1\vec{p}[\,{}^0_1M] = {}^2\vec{p}[\,{}^0_2M] \tag{2.34} $$

and

$$ {}^1\vec{n}[\,{}^0_1M] = \pm\ {}^2\vec{n}[\,{}^0_2M] \tag{2.35} $$

where,

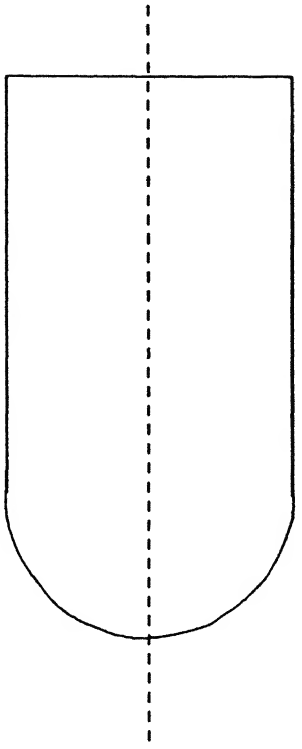
${}^1\vec{p}$ - Point P with reference to the cutter coordinate system S_1.

${}^2\vec{p}$ - Point P with reference to the blank coordinate system S_2.

${}^1\vec{n}_{P,1}$ - Unit normal vector of the cutter surface $\sum_1$ at point P with reference to coordinate system S_1.

${}^1\vec{n}_{P,2}$ - Unit normal vector of the blank surface $\sum_2$ at point P with reference to coordinate system S_2.

[1] We will see in subsequent discussion that it may not be easy in all cases.



Profile of the Cutter

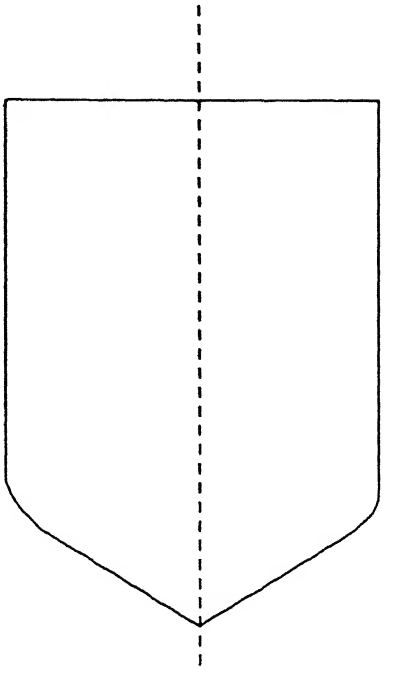
Is arc tangential?	d	r	a	b	h_1	e	f	h
Yes	25.0	12.5	0.0	0.0	57.5	0.0	12.5	12.5

Input Data to Symbolic Manipulator

Range of u	X_c	Y_c	Z_c
-	-	-	-
$0.0 \leq u \leq 19.635$	$12.5\cos[0.08u - 1.571]$	0.0	$12.5\sin[0.08u - 1.571]$
-	-	-	-
$19.635 \leq u \leq 77.135$	12.5	0.0	$u - 7.135$

Output from Symbolic Manipulator

Figure 2.3: Ball Nose End Mill Cutter



Profile of the Cutter

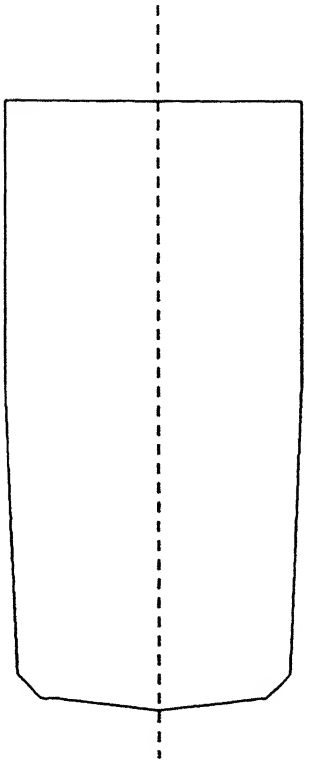
Is arc tangential?	d	r	a	b	h_1	e	f	h
Yes	25.0	4.0	45.0	0.0	55.843	8.5	14.157	5.561

Input Data to Symbolic Manipulator

Range of u	X_c	Y_c	Z_c
$0.0 \leq u \leq 16.021$	0.707u	0.0	0.707u
$16.021 \leq u \leq 19.162$	$4\cos[0.25(u-16.021)$ -0.785]+0.508	0.0	$4\sin[0.25(u-16.021)$ -0.785]+14.157
-	-	-	-
$19.162 \leq u \leq 75.006$	12.5	0.0	$u-5.006$

Output from Symbolic Manipulator

Figure 2.4: 90 deg. Tool



Profile of the Cutter

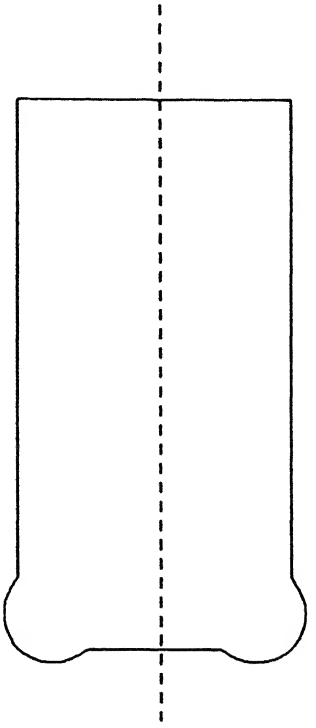
Is arc tangential?	d	r	a	b	h_1	e	f	h
Yes	25.0	4.0	10.0	5.0	50.0	8.782	5.61	20.0

Input Data to Symbolic Manipulator

Range of u	X_c	Y_c	Z_c
$0.0 \leq u \leq 9.624$	0.985u	0.0	0.174u
$9.624 \leq u \leq 14.860$	$4\cos[0.25(u - 9.264)$ -1.396]+8.783	0.0	$4\cos[0.25(u - 9.264)$ -1.396]+5.61
$14.86 \leq u \leq 29.654$	0.087(u-29.654)+14.057	0.0	0.996(u-29.654)+5.61
$29.654 \leq u \leq 79.654$	14.057	0.0	$u - 9.654$

Output from Symbolic Manipulator

Figure 2.5: Side and Bottom Angle Cutter



Profile of the Cutter

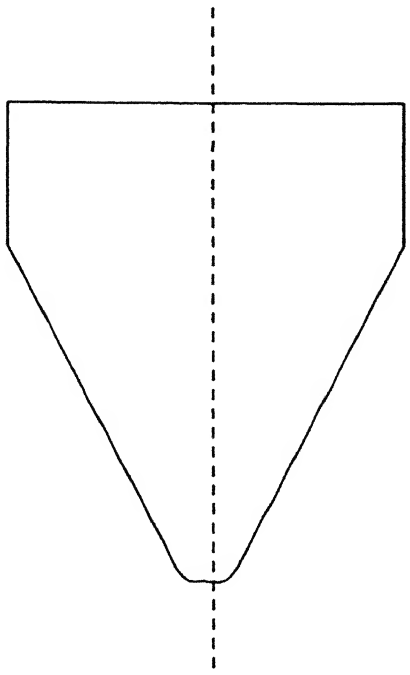
Is arc tangential?	d	r	a	b	h_1	e	f	h
No	25.0	6.0	0.0	0.0	61.528	8.5	4.0	8.472

Input Data to Symbolic Manipulator

Range of u	X_c	Y_c	Z_c
$0.0 \leq u \leq 4.028$	u	0.0	0.0
$4.028 \leq u \leq 23.545$	6 cos[0.167(u − 4.028) -2.412]+8.5	0.0	6 cos[0.167(u − 4.028) -2.412]+4.0
-	-	-	-
$23.545 \leq u \leq 85.073$	12.5	0.0	u − 15.073

Output from Symbolic Manipulator

Figure 2.6: Non-Fillet Type End Mill



Profile of the Cutter

Is arc tangential?	d	r	a	b	h_1	e	f	h
Yes	5.181	4.0	0.0	24.141	20.0	0.0	4.0	50.0

Input Data to Symbolic Manipulator

Range of u	X_c	Y_c	Z_c
-	-	-	-
$0.0 \leq u \leq 4.608$	$4\cos[0.025u - 1.571]$	0.0	$4\cos[0.025u - 1.571]$
$4.608 \leq u \leq 56.809$	-0.837(u - 56.809) + 25.0	0.0	0.548(u-56.809)+50.0
$56.809 \leq u \leq 76.809$	25.0	0.0	u - 6.809

Output from Symbolic Manipulator

Figure 2.7: Conical Cutter

$[\,{}^0_1M]$ - Homogeneous transformation matrix for transforming a vector from the coordinate system S_1 to S_0.

$[\,{}^0_2M]$ - Homogeneous transformation matrix for transforming a vector from the coordinate system S_2 to S_0.

The surface $\sum_2$ is the boundary of the *swept volume* of the cutter surface $\sum_1$, which is nothing but the envelope to the various instances of the cutter surface. We observe that at any given instance, only a single profile is cutting the blank surface[2]. We also observe that this particular profile (out of potentially infinite profiles) is the one where the common normal (the surface are assumed to be conjugate) of the surfaces $\sum_1$ and $\sum_2$ is orthogonal to the relative velocity vector between the cutter and the blank (Chakraborthy and Dhande, 1977). This is what is technically known as *condition of cutting* or the *condition of contact* or the *condition of envelope*. Mathematically, it can be expressed as,

$$ {}^0\vec{n}_{P,1} \cdot {}^0\vec{V}_{P,12} = 0 \tag{2.36} $$

and

$$ {}^0\vec{n}_{P,2} \cdot {}^0\vec{V}_{P,12} = 0 \tag{2.37} $$

where,

${}^0\vec{n}_{P,1}$ - Unit normal vector of the cutter surface $\sum_1$ at point P with reference to coordinate system S_0.

${}^0\vec{n}_{P,2}$ - Unit normal vector of the blank surface $\sum_2$ at point P with reference to coordinate system S_0.

${}^0\vec{V}_{P,12}$ - Relative velocity between the tool and blank with reference to coordinate system S_0.

The condition of contact, being a dot-product between two vectors, is a scalar equation relating parameters u, v and t. Thus, at any given instant of time t, the condition of contact reduces to a relation between u and v. For any value of u within

[2]Later, a case will be seen, where this is not the case. But for most cases this holds true.

its feasible range, one can find the corresponding value of the parameter v. Thus, one can locate points of surface $\sum_1$ which define the surface $\sum_2$. The corresponding points of the surface $\sum_2$ can be obtained using the equation,

$$^2\vec{p} = {}^1\vec{p}[\,{}^0_1M][\,{}^2_0M] \tag{2.38}$$

2.3.1 The Algorithm

In this section, we give the algorithm to find the machined surface $\sum_2$. It has been programmed in the symbolic manipulator developed in this work. It can be stated as follows :

1. Define the position vector $^1\vec{p}$ of a generic point P on the cutter surface $\sum_1$ as a biparametric surface in u and v such that

$$^1\vec{p}(u,v) = [x_1(u,v)\ y_1(u,v)\ z_1(u,v)\ 1] \tag{2.39}$$

2. Define the twelve parameters of motion a_1, b_1, c_1, θ_1, ϕ_1, ψ_1, a_2, b_2, c_2, θ_2, ϕ_2 *and* ψ_2 in terms of t.

3. Calculate the matrices $[\,{}^0_1M]$ and $[\,{}^0_1\dot{M}]$ as follows :

$$[\,{}^0_1M] = [\,{}^0_1R_x][\,{}^0_1R_y][\,{}^0_1R_z][\,{}^0_1T] \tag{2.40}$$

where,

$$[\,{}^0_1R_x] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_1 & \sin\theta_1 & 0 \\ 0 & -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[\,{}^0_1R_y] = \begin{bmatrix} \cos\phi_1 & 0 & -\sin\phi_1 & 0 \\ 0 & 1 & 0 & 0 \\ \sin\phi_1 & 0 & \cos\phi_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[\,{}^0_1R_z] = \begin{bmatrix} \cos\psi_1 & \sin\psi_1 & 0 & 0 \\ -\sin\psi_1 & \cos\psi_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[\,{}^{0}_{1}T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a_1 & b_1 & c_1 & 1 \end{bmatrix}$$

Matrix $[\,{}^{0}_{1}\dot{M}]$ is the time-differential of matrix $[\,{}^{0}_{1}M]$.

4. Calculate the matrices $[\,{}^{0}_{2}M]$ and $[\,{}^{0}_{2}\dot{M}]$ as follows :

$$[\,{}^{0}_{2}M] = [\,{}^{0}_{2}R_x][\,{}^{0}_{2}R_y][\,{}^{0}_{2}R_z][\,{}^{0}_{2}T] \tag{2.41}$$

where,

$$[\,{}^{0}_{2}R_x] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_2 & \sin\theta_2 & 0 \\ 0 & -\sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[\,{}^{0}_{2}R_y] = \begin{bmatrix} \cos\phi_2 & 0 & -\sin\phi_2 & 0 \\ 0 & 1 & 0 & 0 \\ \sin\phi_2 & 0 & \cos\phi_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[\,{}^{0}_{2}R_z] = \begin{bmatrix} \cos\psi_2 & \sin\psi_2 & 0 & 0 \\ -\sin\psi_2 & \cos\psi_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[\,{}^{0}_{2}T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a_2 & b_2 & c_2 & 1 \end{bmatrix}$$

$$[{}^{2}_{0}M] = [{}^{0}_{2}M]^{-1}$$

Matrix $[\,{}^{0}_{2}\dot{M}]$ is the time-differential of matrix $[\,{}^{0}_{2}M]$.

5. Derive the position vector ${}^{2}\vec{p}$.

$${}^{2}\vec{p} = [x_2 \;\; y_2 \;\; z_2 \;\; 1]$$

$${}^{2}\vec{p} = {}^{1}\vec{p}[{}^{0}_{1}M][{}^{2}_{0}M] \tag{2.42}$$

6. Derive the normal vector ${}^{1}\vec{N}_{P,1}$.

$$ {}^1\vec{N}_{P,1} = \frac{\partial {}^1\vec{p}}{\partial u} \times \frac{\partial {}^1\vec{p}}{\partial v} \tag{2.43} $$

where,

$$ \frac{\partial {}^1\vec{p}}{\partial u} \quad and \quad \frac{\partial {}^1\vec{p}}{\partial v} $$

are the partial derivatives of ${}^1\vec{p}$ with respect to u and v respectively. Therefore,

$$ \frac{\partial {}^1\vec{p}}{\partial u} = [\frac{\partial x_1}{\partial u} \; \frac{\partial y_1}{\partial u} \; \frac{\partial z_1}{\partial u} \; 0] $$

$$ \frac{\partial {}^1\vec{p}}{\partial v} = [\frac{\partial x_1}{\partial v} \; \frac{\partial y_1}{\partial v} \; \frac{\partial z_1}{\partial v} \; 0] $$

When the vectors

$$ \frac{\partial {}^1\vec{p}}{\partial u} \quad and \quad \frac{\partial {}^1\vec{p}}{\partial v} $$

are expressed in cartesian coordinates, the normal vector ${}^1\vec{N}_{P,1}$ is defined as :

$$ {}^1\vec{N}_{P,1} = \begin{vmatrix} i & j & k \\ \frac{\partial x_1}{\partial u} & \frac{\partial y_1}{\partial u} & \frac{\partial z_1}{\partial u} \\ \frac{\partial x_1}{\partial v} & \frac{\partial y_1}{\partial v} & \frac{\partial z_1}{\partial v} \end{vmatrix} $$

The same can be redefined when the vectors

$$ \frac{\partial {}^1\vec{p}}{\partial u} \quad and \quad \frac{\partial {}^1\vec{p}}{\partial v} $$

are expressed in homogeneous coordinates as :

$$ {}^1\vec{N}_{P,1} = \begin{vmatrix} i & j & k & l \\ \frac{\partial x_1}{\partial u} & \frac{\partial y_1}{\partial u} & \frac{\partial z_1}{\partial u} & 0 \\ \frac{\partial x_1}{\partial v} & \frac{\partial y_1}{\partial v} & \frac{\partial z_1}{\partial v} & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} $$

7. Calculate the unit normal vector[3] ${}^1\vec{n}_{P,1}$.

$$ {}^1\vec{n}_{P,1} = \frac{{}^1\vec{N}_{P,1}}{\| {}^1\vec{N}_{P,1} \|}. \tag{2.44} $$

[3]Actually this step may be skipped.

where

$$\| {}^1\vec{N}_{P,1} \|$$

is the magnitude of

$${}^1\vec{N}_{P,1}$$

.

8. Calculate the unit normal vector[4]

$${}^1\vec{n}_{P,1}$$

in the global coordinate system.

$${}^0\vec{n}_{P,1} = {}^1\vec{n}_{P,1} \, [\, {}^0_1M] \tag{2.45}$$

9. Calculate the relative velocity vector ${}^0\vec{V}_{P,12}$.

$${}^0\vec{V}_{P,12} = {}^0\vec{V}_{P,2} - {}^0\vec{V}_{P,1} \tag{2.46}$$

where,

$${}^0\vec{V}_{P,1} = {}^1\vec{p}[\, {}^0_1\dot{M}] \; and \; {}^0\vec{V}_{P,2} = {}^2\vec{p}[\, {}^0_2\dot{M}]$$

10. Reduce symbolically the condition of contact

$${}^0\vec{n}_{P,1} \cdot {}^0\vec{V}_{P,12} = 0 \tag{2.47}$$

11. ${}^2\vec{p}$ obtained in step 5 will be a function of u,v and t. Use the condition of contact obtained in step 10 to eliminate either u or v. This will yield the surface $\sum_2$.

[4]This will not be unit normal if the earlier step is skipped. But that makes no difference as we will observe later.

2.4 Computational Model

As observed earlier, it may not always be possible to symbolically eliminate a variable between equations (2.42) and (2.47). Infact, even for the most simple kinematics the equations become very complex. In such circumstances we revert to numerical methods.

As explained earlier, the generic cutter is modeled as a biparametric , axisymmetric surface. Equation (2.42) gives us the cutter surface at any time instant t in the coordinate reference frame S_2. Hence it is a function of u, v and t. Similarly, equation (2.47) gives the condition of contact which the cutter surface must satisfy at any time. This condition of contact depends upon the cutter geometry and the relative kinematics between the cutter and blank surfaces. A point worth noticing is that the condition of contact is independent of the coordinate reference frame chosen. This does not mean that the equation for condition of contact is same for all coordinate systems, but only that the solution value of v is same in all coordinate systems. This is because neither of the transformations, viz. translation or rotation changes the angles between the vectors ${}^0\vec{n}_{P,1}$ and ${}^0\vec{V}_{P,12}$. This condition of contact is also a function of u, v and t. Now , to get the surface $\sum_2$ we have to eliminate either of u or v. It can be easily observed that it would be most convenient to eliminate v. We also make the following observations :

- For a given time instant t' and parametric value u' of u, there are two values of v, say v_1 and v_2 which satisfy the condition of contact and such that

$$v_1 = v_2 + \pi$$

- For such v_1, v_2 and t' and for *all* u the points ${}^2\vec{p}(u, v_1, t')$ and ${}^2\vec{p}(u, v_2, t')$ are on the surface $\sum_2$.

All the corresponding points of the surface $\sum_2$ are then calculated and stored in an array and rendered later.

2.5 Some Observations

Till now we were tacitly assuming that there will be only a single profile of the cutter which will be orthogonal to the relative velocity vector. This is not the case. Even if this were the case, there are other situations in which this algorithm fails. The failure does not imply any fault in the symbolic equations derived as such, but that such equations cannot always be rendered automatically. In such places the user intervention becomes necessary. And the user needs to be a C - programmer well versed in the use of graphics package employed (in this case - STARBASE). In the following discussion we give some cases where user intervention becomes necessary :

1) Not all types of kinematics are allowed : There are many simple cases of motion of the cutter and blank where the present approach fails. One such case is the drilling motion. Assuming that the blank is stationary, consider the case when the cutter is moving along negative Z direction. Such a motion, in practice, will produce a hole in the blank. But in the present case we observe that there are not two but many (potentially infinite) profiles satisfying the condition of contact and thereby becoming eligible for inclusion in the machined surface. Even if we somehow show all the profiles the problem remains that how to connect each profile at a particular time instant with that of the next instant. And there is the problem of self intersection of the cutter because the previously cut portion of the blank by the tip of the cutter is cut later by the tail piece of the cutter. In such cases it becomes difficult (even in *adhoc* fashion) to determine which portion of the blank is to be retained and which removed.

2) Crossing of the cutter path is not allowed : Even though ,logically, this is just another instance of the previous case, it deserves special attention. In cases of this type, everything goes on fine until ,suddenly, the cutter starts cutting the portion cut earlier. As earlier, here too it is not obvious what the final shape of the machined surface should be. Though there are some algorithms which calculate surface-surface intersection, it is not clear how much they will be useful in the present case.

3) The cutter has to be axisymetric : Throughout the work, it has been

assumed that the cutter is axisymmetric. Infact, this fact has been used in the symbolic definition of the cutter, where only one half of the profile is defined and the rest of the cutter is developed by rotating this profile around the Z axis. But in practice this may not be the case, though even for most non-axisymmetric cutters we can argue that they are generally rotating, hence effectively forming an axisymmetric cutter surface. It should be noted that a cutter, such as a helically swept one, can be defined and used, but the rendition of final machined surface , is not easy. The problem here is that a helical cutter may *screw* its way into the blank and such cases are very tough to handle.

4)The cutter motion is assumed continuous : Previously we said that the surface is obtained by joining different instances of cutter profile at different time instants such that the profile is orthogonal to the relative velocity vector. But in practice what generally happens is that a portion of the blank is machined and then the cutter is shifted to another portion and the machining started again. In such cases the cutter path (or the machined surface) is a set of unconnected surfaces instead of a single, continuous surface. How are such cases handled?

In this work such cases are ignored. Such cases can be handled using interval functions. But we see that such cases can also be rendered by taking different runs for each interval of the function. Incidentally, this is the flaw (the symbolic manipulator developed in this work does allow definitions of interval functions. But how to interpret them is left to the programmer[5].) of the symbolic manipulator rather than that of the approach.

5)The blank is assumed to be present everywhere : One of the assumptions which was made is that the cutter never runs out of blank, i.e the cutter is moving in a blank which practically extends to infinity in all directions. Obviously this is not the case in general. In many cases it becomes important to have an initial shape for the blank (All blanks have some initial shape) because in such cases we are interested in the piece that is left rather than the groove which is cut by the cutter. One such example is that of a gear. Though it is possible to visualize what the gear would look

[5]For more information, kindly refer to the appendix-A given at the end of the report

like, it is taxing on the user's imagination.

And also in many situations we would like to have an initial shape for the blank just to see how many different workpieces could be made out of the given piece of material. If such a mechanism is provided, then the user could do the machining with different initial positions and orientations. One can then augment the basic simulator with one of the numerous algorithms for optimally fitting the workpieces into the given blank piece thereby minimizing the wastage of blank material. This is one of the most important criteria for the tool designer.

Chapter 3

Symbolic Manipulator

3.1 Role of symbolic manipulator

In this section we will discuss what a symbolic manipulator is, and what it does. After the initial history, we proceed to explain why at all it is needed and its advantages over conventional computing.

3.1.1 Symbolic Manipulation – What is it?

Solving problems with symbols substituted for actual entities is known as symbolic manipulation. In the present context, it means doing computation on variables which may or may not be bound to some particular numerical values.

3.1.2 History and development

Human species is unique in this respect that it is the only one to posess this capability to a substantial degree. Computers, which are replacing man in almost every field, have traditionally been used for numerical computation so much so that they have come to be known as *number crunchers*. But recently, with the advances in hardware and software technologies, it has been possible to use computers in symbolic computation. The idea, though, is not new. It dates back to 1840's, when Lady Ada Lovelace[1] first suggested it.

[1]Lady Ada Lovelace was a patron of Charles Babbage, who is credited with the development of world's first computer

Initially the symbolic manipulator packages were written for specific fields. But with the breakthrough in hardware technologies, we have a choice among a wide spectrum of computers, from P.Cs to mainframes, with computational speeds touching hundreds of Mflops(Mflops stands for Million floating point operations per second) and memories, correspondingly, running into several Giga bytes (1 Giga byte = 2^{30} bytes). With this, it has now become possible to develop packages which are general in nature and work on various platforms. *MACSYMA* and *MathCAD* are two excellent examples.

The symbolic manipulation systems are capable of performing a wide variety of operations. They can do not only polynomial, but also trigonometric and matrix algebra. They can solve a system of equations and find roots of an expression. They can also solve problems involving differentiation and integration. Some packages like *MACSYMA* can also find numerical solutions whenever needed.

3.1.3 Why a symbolic manipulator?

Today we are living in an age where graphical interfaces and visual computing are becoming increasingly common. The main reason behind their success is that they give more insight into the problem than any other mode of depicting information. One of the goals of any computation is conveying maximum information in minimum period of time. An algebraic equation, for example, conveys more information to a scientist or mathematician than the enumeration of many points of that equation.

But this is not all there is to symbolic manipulator. Symbolic manipulator maintains the generality of computation. Therefore, it is particularly well suited for developing generic algorithms. Secondly, the precision of numerical systems is hardware dependent. The rounding and truncation may produce a cumulative effect and produce erroneous results. With symbolic computation, the rounding and truncation can be deferred till the final steps, thereby minimizing the errors. Symbolic manipulators can deal with zero, infinity and other such exceptional conditions more elegantly. Many times it is possible to use the intermediate computations for entirely

different purposes[2]. If it is desired to store the output, the storage overhead for numerical result is many times more than the corresponding symbolic one. Moreover, the numerical results are inextricably interwoven with the idiosyncracies of the data structures and numerical methods used, which is not the case with symbolic results.

3.2 Software Design and Implementation

The symbolic manipulator is the heart of the present work. It would, therefore be useful to study its inner workings. In this section we will be doing just that.

Conceptually the symbolic manipulator consists of the following stages :

- Parsing
- Symbol table construction
- Symbol substitution and function evaluation
- Expression minimization

It should be noted that the first two stages are done in sequence for all the declarations and last two of the above steps are for each individual declaration as would become clearer later.

Fig (3.1) gives a pictorial representation of the basic relationships among different modules. The actual relationships are however very complex.

Parsing: In parsing stage the input is read one token at a time. For this purpose the parser calls the token recognizing routine repetitively and tries to fit the input into a predefined format called the grammar of the symbolic manipulator. The grammar defines what kind of input is expected and accepted by the symbolic manipulator.[3] Most of the grammar resembles closely the C-grammar, so that there will not be much problem in using it.

The main constructs recognized by the symbolic manipulator are:

[2]One such case is discussed in chapter 4

[3]A formal specification of the symbolic manipulator grammar is given in Appendix A

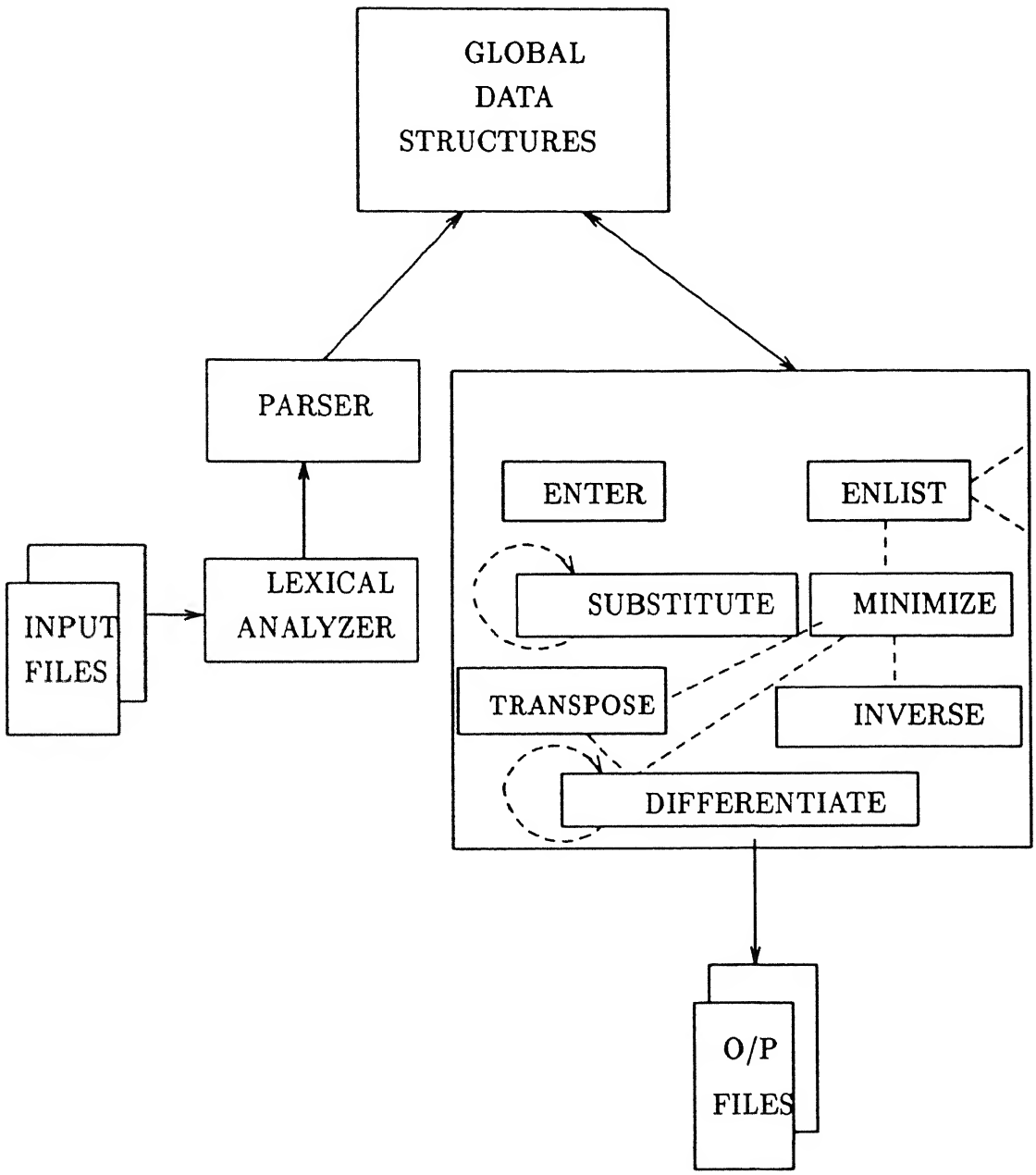


Figure 3.1: The Symbolic Manipulator

- Identifier declaration and initialization
- Matrix declaration and initialization
- Function declaration
- Function call

The initialization for functions and identifiers is done by a single expression and for matrices it is a comma separated list of expressions. An expression may contain, besides identifiers and constants, matrix and function names as well as function calls all connected by allowed operators.

The parser constructs a parse tree in memory by using a dynamic array of tokens. This array is called *Table*. Each token contains information about the present entity stored there and the indices to left and right subtrees. The structure *Token* is defined as follows:

```
struct Token{
double value;
char string[20];
int type;
int ileft;
int iright;
int valid;
int op;
};
```

The field *string* holds the name of the token. The *value* field holds the numerical value if it is a constant. Field *type* holds the the type of the token, i.e. whether it is an identifier ,a constant or a matrix and so on. The fields *ileft* and *iright* are indices into the array where the left and the right subtrees may be found. Field *valid* was provided for the purpose of garbage collection. The last field *op* is the ASCII value

of the operators like +,-,/ and so on and predefined values for operators like sin, cos etc.

The root of the parse tree is left in the global variable *out*. Once the parsing is over, the tree is copied from *Table* to a similar array *safe*. The *Table* is then freed. The reason for this movement of the tree is that during computation, each individual declaration is brought into *Table* and manipulated. That leaves a lot of *Tokens* which are no longer used. To collect this garbage, the most effective way is to free the whole *Table*, which is possible only if the rest of the declarations are stored in a safe place.

Symbol Table Construction: Once the parse tree is in *safe*, a routine *enter* is called which, for each symbol encountered, makes an entry in the symbol table. In this process the latest definition overrides the previous one. It is also worth mentioning here that there should be no conflict of names even among different types of identifiers, e.g. between a matrix name and a function name.

The symbol table is an array of structures called *SYM_TAB* which looks like:

```
struct SYM_TAB{
char ident[20];
int defined;
int type;
int dep_list;
int order;
int definition;
}
```

The field *ident* stores the name of the token, if any. Field *defined* is non-zero if the corresponding symbol is defined and zero otherwise. The type of the token is given by *type*. For functions, *dep_list* gives the variables on which the function might depend. The field *order* is exclusively for matrices and gives its order(through the

index into the *safe*) and *definition* is the index into the array where the definition of the current symbol may be found.

The symbol table not only provides a convenient way to access information, but also associates a unique key (the slot number) with each symbol. This key can be used to develop more efficient algorithms, explained later.

Substitution and Evaluation: After the symbol table is constructed the tree is again traversed and each declaration is copied into the *Table* and processed, one at a time. Once a declaration is in *Table*, its symbols are substituted by their corresponding definitions accessed from the *safe* through the symbol table. The substitution is done recursively, i.e. a symbol is first substituted *fully* before proceeding further. This unfortunately has a serious drawback (though it in no way affects the performance for the present purpose) that it reduces the capabilities of the symbolic manipulator somewhat. Later we discuss how this can be remedied with minimal changes in code. The problem is dealt with in chapter5. Here it would suffice to say that by cutting on this capability we have customized the symbolic manipulator to the present need.

In the evaluation phase a function *evaluate* is invoked with current declaration (that which is in the *Table* at present) as its argument. This function as its name suggests, evaluates the function calls embedded in the current declaration.

E.g. suppose that the current declaration is:

```
z = __differentiate(x){x^2.0;}*__differentiate(y){y^3.0;};
```

then evaluate would further invoke the *differentiate* routine with arguments x and $x^{2.0}$ (actually the indices into the *Table*) followed by y and $y^{3.0}$ and after that return.

Minimization: After *evaluation*, the next step is minimization. There are several kinds of minimization performed. The major ones are:

- Cancellation of terms with opposite(same) terms in addition(subtraction).

- Cancellation of identical terms in division.
- Cancellation of nested *inverse* operators like $\sin(\arcsin(x + \ldots))$ is reduced to $x + \ldots$.
- Cancellation of *reverse* operators in multiplication, like, $\sin(x) * \csc(x)$ is reduced to 1.0.

and so on.

For this purpose, the operators are divided into the following groups:

- The addition/subtraction group.
- The multiplication/division group.
- The trigonometric group.

The trigonometric group triggers special treatment functions for trigonometric expressions and after that they also come into one of the first two groups. So, in effect, there are only two major groups.

In the addition/subtraction group all expressions are treated as consisting of tokens connected by operators *binary* $+$ *or* $-$ or *unary* $+$ *or* $-$. Sub-expressions consisting of other operators like *, / etc. are also assumed to be a single albeit compound token.

For this purpose a recursive function *enlist* is developed which is passed the root of a subtree. Depending upon the current operator at the root of the subtree, *enlist* decides which of the three groups the sub-expression falls into. Thus deciding on the group, it recursively calls itself on the left and right subtrees.

At the bottom level of such call, i.e. when *enlist* reaches the leaves, *enlist* enters information about each of the leaves in an array of structures, called *listnodes*. A *listnode* looks like:

```
int key;
int index;
int valid;
int direct;
}
```

The field *key* associates a key to each token which can be effectively used to sort them in case more efficient algorithms for minimization are used. The field *index* is the index into the *Table* where the token can be found. Field *valid* field is used in minimization to mark the cancelled tokens as invalid. The last field, *direct*, was used in the earlier stages of implementation and is no longer used.

For each sub-expression we make two lists depending upon the group it belongs to. The addition/subtraction group has a plus-list and a minus-list. Similarly, the multiplication/division group has a num-list and den-list. These lists are stored in the same global array of structures *listnodes*. The array is an array of *listnodes* and is called *list*. The information regarding where each list starts and ends is passed across functions through pointer to structure *listhead*. A *listhead* consists of the following fields:

```
struct listhead{
int l1;
int l2;
int type;
}
```

Field $l1$ is an index into array *list* and indicates where the plus-list or num-list starts. In a similar way $l2$ indicates where the other list starts. The end of the currently active (at any given time there are more than one set of lists except of

course when *enlist* is at the top level) minus or den list is stored in a global variable. Finally the field *type* is the indicator of the group the token belongs to.

The bottom level *enlist* passes the pointer to this structure *listhead* to the routine *minimize* which does many minimizations and then *enlist* makes a new *listhead*, fills up the minimized information and returns it to the caller function. The caller function then extracts information from its left and right *listheads* merges them keeping in mind what types are the left and right *listheads* and proceeds further similar to the lower level *enlist* call. This goes on till the top level is reached and the minimization is over (note that the minimization is over as far as this symbolic manipulator is concerned, it may not be, however, complete). Then the current minimized declaration is copied back back into the *safe* and the symbol table entry is changed to reflect the changes.

Here it may be observed that a lot of time and memory may be saved using *forward* declaration of the symbols used, i.e. defining a symbol before it is used. However, this is not imperative.

3.3 Some Pitfalls

It is well known that any symbolic manipulator makes excessive demands on the system memory, as the intermediate expressions can become arbitrarily long. The symbolic manipulator developed in this work is no exception to this, though an attempt is made to reduce it by cutting on the generality of the symbolic manipulator.

The design of the symbolic manipulator presented in the previous section is quite efficient with regards memory and run time[4]. But there are some cases, which had to be handled separately.

Except for the functions *inverse* and *taninv*, all other functions are robust. Robust in the sense that they can be used wherever it logically makes sense. Finding the cross product of two matrices makes sense, but crossing two functions doesn't, for example.

[4]For further improvements, refer to Chapter5

The reason for *inverse* being *fragile*[5] is that computing the inverse of a matrix symbolically is heavily taxing on the memory. The algorithm *inverse*, therefore, periodically flushes the *Table* in order to collect garbage. Hence, if the function call *inverse* occurs as on of the terms in an expression, then during the evaluation of *inverse* the rest of the expression is also flushed and lost. That is the reason why matrix inversions need to be done separately.

The case of the function *taninv* is different. *taninv* is similar to the standard C-library function, *atan*2, which takes two arguments, the Y and the X coordinate values and returns the corresponding angle in the interval $[-\pi,+\pi]$. The need of *taninv* was realized in the final stages of the work, and hence only a minimal functionality was provided. The peculiarity is that *taninv* expects both its arguments to be computable at compile time. This however makes no difference present context, as they are indeed computable wherever it is needed in the present work.

Another source of bugs might be the function declaration construct of the symbolic manipulator. In present application, nothing is to be gained by using it. It was provided for future extensions. As any usage of this construct can be modeled by other constructs, it was seldom tested thoroughly.

3.4 Some Examples

We illustrate the workings of the symbolic manipulator with some examples. These examples serve to show only some salient features of the symbolic manipulator and by no means are exhaustive.

```
/*INPUT1*/

y = __differentiate(x){x^x + log(x);}- 2.0*x;

z = (cos(acos(y)) + sin(y)*cosec(y))*tan(t)/(sec(u)*tan(t)*cos(u));
```

[5] *Robust* and *Fragile* are terms borrowed from latex manual

```
matrix test[2.0,2.0]={p,log(p),2.3,4.9};

testinv = __inverse(){test;};

/*OUTPUT1*/

y=((x)^(x))*(1.000000+(1.000000)*(log(x)))+(1.000000)/(x)-
(2.000000)*(x);

z=((x)^(x))*(1.000000+(1.000000)*(log(x)))+(1.000000
)/(x)+1.000000-(2.000000)*(x);

test[2.000000,2.000000]{p,log(p),2.300000,4.900000 };

testinv=mult[2.000000,2.000000]{(-9.800000)/((p)*(-4.900000)+
(log(p))* (2.300000)+(2.300000)*(log(p))+(4.900000)*(-(p))),
((log(p))*(2.000000)) /((p)*(-4.900000)+(log(p))*(2.300000)+
(2.300000)*(log(p))+(4.900000)* (-(p))),(4.600000)/((p)*(-4.900000)
+(log(p))*(2.300000)+(2.300000)* (log(p))+(4.900000)*(-(p))),
((-(p))*(2.000000))/((p)*(-4.900000)+(log(p)) *(2.300000)+
(2.300000)*(log(p))+(4.900000)*(-(p)))};

/*ADDED INPUT*/

p = 3.246;

unit = __mmult(){test*testinv;};

/*OUTPUT*/
test[2.000000,2.000000]{3.246000,1.177423,2.300000,4.900000};
```

```
testinv[2.000000,2.000000]{0.371287,-0.089217,-0.174278,0.245959 };

unit=mult[2.000000,2.000000]{1.000000,0.000000,0.000000,1.000000};
```

Chapter 4

Graphical Simulation

In the previous two chapters, we saw the methodology to be followed for obtaining the symbolic equations of the machined surface and the implementation of the symbolic manipulator. But the main motive of the work is to render the surfaces graphically. In the following section we discuss how it is achieved.

4.1 Surface Rendering

There are many ways a surface can be rendered. The major ones are :

- Wireframe
- Filled Polygons
- Spline surfaces

The first approach, though simple and fast, is not suitable for the present need. The surfaces are of free form, and except for very simple ones it becomes difficult to visualize them with this model. The last approach gives a very realistic rendering of the surfaces. But the calculation of splines dynamically is a very computation intensive operation and this fact makes them unfit for our purpose.

The second approach, coupled with shading techniques (Goraud in the present case), is ideally suited for the current application. In this approach, the surface is approximated by polygons. In order to reduce the faceted appearance of the surface

and get a smoother surface, the intensity is interpolated across polygons. For this purpose normals need to be specified at each vertex of each polygon in the surface.

4.1.1 Calculation of Normals

In general there are two approaches used for calculation of normals depending upon the tradeoff between the smoothness of animation and realism of the rendition.

The first approach, fast but inaccurate, is to calculate the normals at a vertex by taking the cross product of two edges of the polygon incident on that vertex. The shading based on such normals is uneven and a close examination reveals the faceted nature of the surface.

In the second approach, the normal at a vertex is calculated by taking the resultant of all normals due to all the edges incident on a vertex, instead of just two. This gives a much smoother shading to the surface.

In our case, we have the choice of going for an even better rendition by calculating exact normals at each vertex. Since we have the symbolic equations of the cutter surface normals in the S_0 frame of reference, all that is to be done is to transform them to the S_2 frame of reference. This is done by postmultiplying them with the transformation matrix M_0^2 using symbolic manipulator. Having obtained the symbolic normals, we only have to give different values to u, v and t to get the normals at different vertices. The surfaces rendered thus are highly realistic as the photographs illustrate. The animation also does not suffer.

4.2 Animation

The field of computer animation has progressed appreciably during the past decades owing to evolution of special purpose hardware and powerful techniques such as back-face culling, hidden-surface removal and double buffering which has made it possible to render motion realistically.

A manufacturing process is a dynamic process. It is not very impressive to show just the cutter and the machined surfaces. If the kinematics are also shown then the

designer can visualize the motion and change the kinematics appropriately to get the desired surface.

In this work two distinct types of animation may be observed. One is due to the kinematic parameters supplied earlier and is fixed. The other is because of the movement of the observer, which is interactive. The capability of moving the observer enables to view the machining process from different view points. This kind of animation is a standard in graphics and we won't discuss it any further than saying that many options are available for the user.

The first kind of animation, basically, shows the motion of the cutter as seen from the reference frame S_2 and the machined surface as the blank is being cut. This is achieved as follows:

From the symbolic manipulator we already have the cutter surface as a function of u, v and t. u and v are the parameters specifying the surface and t is the time. Stepping t between two instants of time with the required value of time step, we obtain the position of the cutter at the corresponding time as guided by the kinematic parameters.

Now, to show the machined surface we compute the cutter profile satisfying the condition of contact and store it along with the normals. It should be clear that the machined surface is completely calculated before any output appears on the screen. Only that portion of the surface is displayed which is actually cut till the present moment. That gives the effect of the blank being cut then and there.

4.3 User Interface

The user interface provided is highly customized to the present application. To use the symbolic manipulator for other purpose say, for computing a different algorithm, one has to access it separately.

The menu is textual and the user is prompted to select one of the following four sub-menus (or quit):

- Cutter specifications

- Kinematics
- Graphics
- Run

The *Cutter specifications* menu asks for the values of nine parameters of the generic cutter. The *Kinematics* menu similarly prompts the user for the kinematics of the cutter as well as the blank surfaces. *Graphics* menu controls the rendering aspects such as the background colour , the type of normals to be used for rendering the cutter surface (fast or exact), the value of time step and the like. The final menu, *Run* gives options for compiling, cleanup, running the program etc.

Default values are provided in all appropriate places.

Since it is very difficult to predict beforehand what the exact numerical values should be for the coefficients of various parameters, as many as 25 variable parameters are supported. Any variable parameter which needs to be specified at the run time should have a name of the type *varparamX*, where X is any number between 1 and 25, both included. This limit of 25 on variable parameters is purely arbitrary and can be set to any desirable value by minimal addition of code. These parameters should be used sparingly as it would increase demands on system memory and time.

4.4 Examples

In this section we give three examples, which have been directly tested on HP-TSRX workstations. The output (graphical) is shown in the photographs that follow:

Example:1:

Is arc tangential?	d	r	a	b	h_1	e	f	h
Yes	5.181	4.0	0.0	24.141	20.0	0.0	4.0	50.0

Table 4.1: The Cutter Parameters for the Conical Cutter

Explanation: The cutter is stationary with its axis coinciding with the Y-axis of the global coordinate system. The blank is rotating around X-axis of global coordinate system and at the same time moving along the positive X-axis (of S_0). See Tables(4.1, 4.2 and 4.3).

Example:2:

$$x_1(u,v) = u\cos v$$
$$y_1(u,v) = u\sin v$$
$$z_1(u,v) = au^2 + bu + c$$

Where

$$0 \leq u \leq 5.33903e+01$$
$$0 \leq v \leq 2\pi$$
$$a = 1.68979e-02$$
$$b = 1.02956e-06$$
$$c = 4.91379e+01$$

The Cutter Equations for End Milling Cutter

Explanation: The cutter axis is in the XY plane, making an angle of $\frac{\pi}{12}$ with the positive Y-axis (of S_0 frame of reference) and the tip of the cutter is at a distance of 2.0 units along Y-axis initially.

The cutter is moving away in positive Y direction while maintaining the above configuration. See Table(4.4).

The blank is rotating around X-axis of global coordinate system and at the same time moving along the positive X-axis (of S_0). See Table(4.5).

Example:3:

Explanation: Initially the cutter tip is at a distance of 200.0 units from the origin along positive Y direction and moves sinusoidally along the Y-axis. See Table(4.6).

The blank rotates around X-axis of global coordinate system and at the same time moves along the positive X-axis (of S_0). See Tables(4.7 and 4.8).

a_1	b_1	c_1	θ_1	ϕ_1	ψ_1
0.0	0.0	0.0	$\frac{\pi}{2.0}$	0.0	0.0

Table 4.2: The Cutter Kinematics of the Conical Cutter

a_2	b_2	c_2	θ_2	ϕ_2	ψ_2
$\frac{-50.07*t*0.0125}{2.0*\pi}$	0.0	0.0	t*0.0125	0.0	0.0

Table 4.3: The Blank Kinematics for the Conical Cutter

a_1	b_1	c_1	θ_1	ϕ_1	ψ_1
0.0	2.0+0.11*t	0.0	$\frac{\pi}{2.0}$	0.0	$\frac{\pi}{12.0}$

Table 4.4: The Cutter Kinematics of the End Milling Cutter

a_2	b_2	c_2	θ_2	ϕ_2	ψ_2
$\frac{-104.0*t*0.025}{2.0*\pi}$	0.0	0.0	t*0.025	0.0	0.0

Table 4.5: The Blank Kinematics for the End Milling Cutter

Is arc tangential?	d	r	a	b	h_1	e	f	h
Yes	25.0	4.0	-10.0	-5.0	50.0	8.061	2.64	20.0

Table 4.6: The Cutter Parameters for Bottom Angle Cutter

a_1	b_1	c_1	θ_1	ϕ_1	ψ_1
0.0	100.0*(1.0+cos(4.0*t*$\frac{\pi}{180.0}$))	0.0	$\frac{\pi}{2.0}$	0.0	0.0

Table 4.7: The Cutter Kinematics of the Bottom Angle Cutter

a_2	b_2	c_2	θ_2	ϕ_2	ψ_2
$\frac{-50.0*t*0.0125}{2.0*\pi}$	0.0	0.0	t*0.0125	0.0	0.0

Table 4.8: The Blank Kinematics for the Bottom Angle Cutter

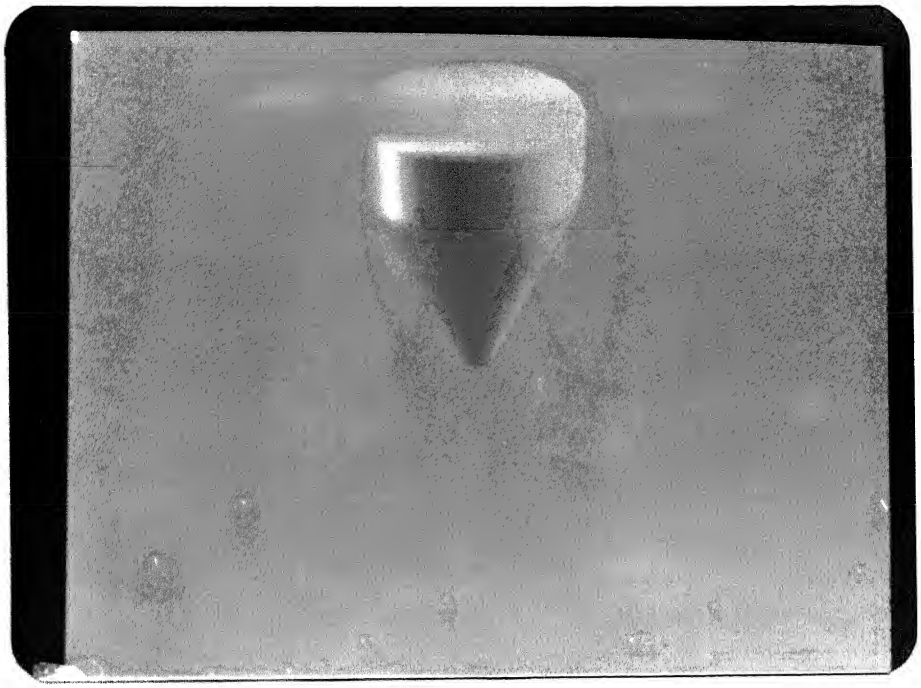


Figure 4.1: The Conical Cutter

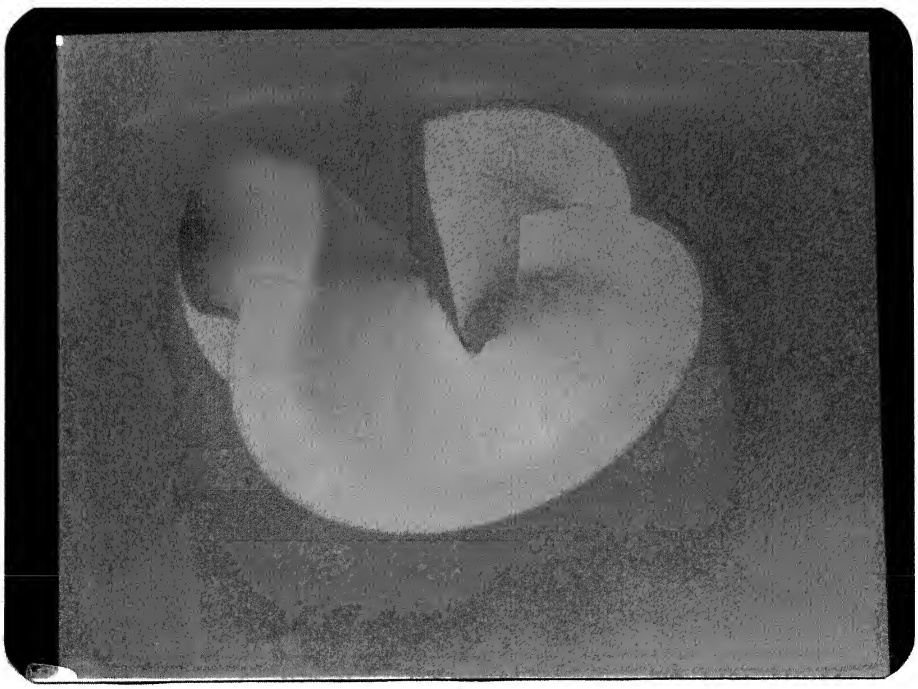


Figure 4.2: The Helical Screw being Machined

Figure 4.3: The Helical Screw

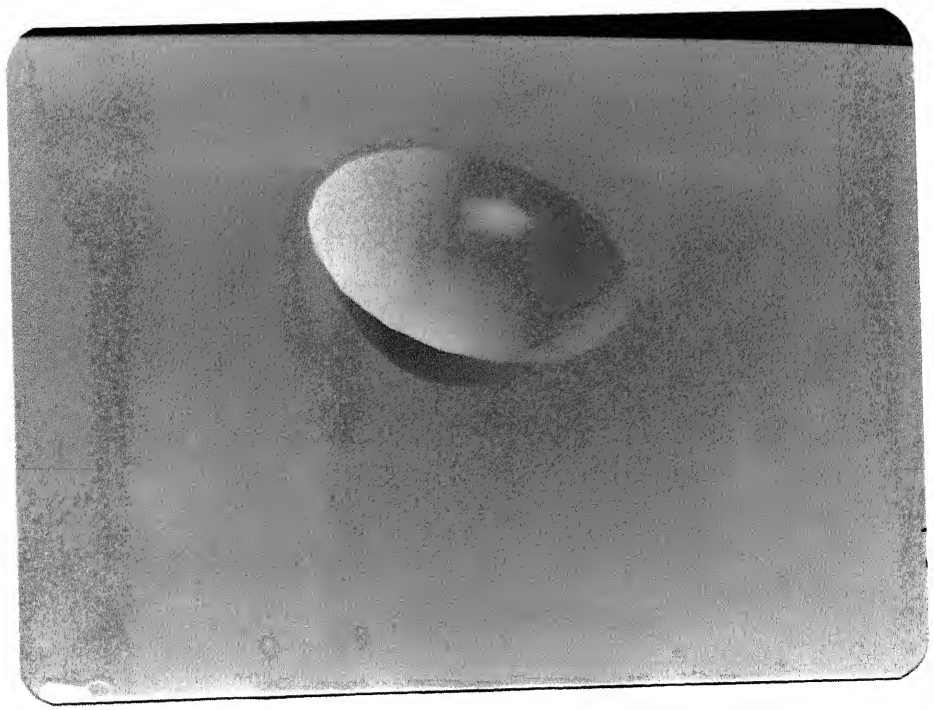


Figure 4.4: The Ball Nosed End Mill Cutter

Figure 4.5: The Spiral Screw being Machined

Figure 4.6: The Spiral Screw Surface

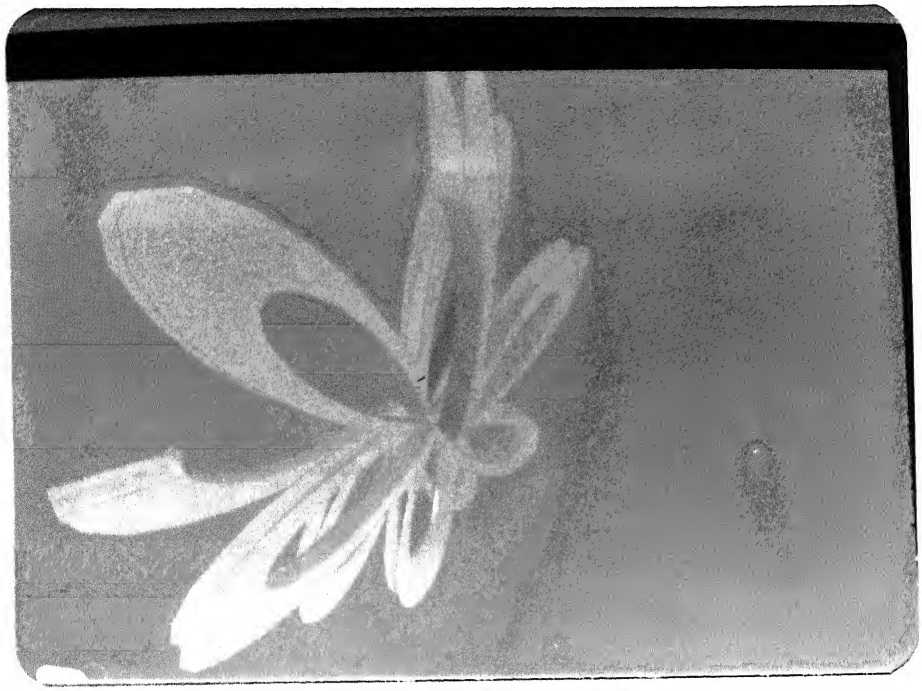


Figure 4.7: The Machining of Star Shaped Surface

Figure 4.8: The Star Shaped Surface

Chapter 5

Conclusions

The present work encompasses a fairly large number of cases encountered in practice. It renders correctly all such surfaces which are cut by axisymmetric cutters such that a portion once cut is not cut again. Even cases in which the cutters are not axisymmetric, can be handled by taking their biparametric equations directly. The cases where a satisfactory rendering could not be obtained are considered in the following section.

5.1 Considerations for Further Work

The main problem with the present approach crops up because of the self intersection of the cutter path. Such cases are not very rare in practice. These cases have to be solved computationally, as it may not always be feasible to solve them symbolically.

To do this one considers the cutter surfaces at a very close interval of time. The *snapshots* so obtained are checked for intersection and the portions which are inside are removed. The resultant surface is still not in an acceptable form as even for very close *snapshots*, the machined surface obtained by the above method might show severe irregularities not to be seen in practice. Such irregularities can then be smoothed out using interpolation near the points of intersection of surfaces.

It may be noted that the intersection need not be with two successive *snapshots* only. It can be with the machined surface cut long before. Therefore in a general

case, one of the surfaces (the machined surface, to be precise), will be an approximate surface based on the points stored for rendition.

All through the work we have assumed that the cutter is fully immersed in the blank which is generally not the case. A further extension could therefore be to define an initial shape for the blank. Then to render the machined surface, we proceed as before and at the end check the developed surface for intersection with the blank surface. The portions which are outside are then removed leaving the actual machined surface.

5.2 Some Techniques and Improvements

In this section we will discuss how to add more functionality to the symbolic manipulator and how to make it more efficient in terms of memory and minimization of expressions.

As promised earlier, we will first look into how substitution of symbols can be made partial. The function *substitute* first substitutes a symbol and then substitutes the previously substituted expression. All that is to be done is to replace this code with an iterative loop whose limit will specify how deep the substitution should be.

The next issue is how to add more functionality to the symbolic manipulator. The manipulation of functions is mainly localized in three routines, namely *reserved*, which checks whether a function is a procedure supplied with the symbolic manipulator, *evaluate*, which on every occurence of a function name, calls the corresponding module in the program and last of all *Display* (and its clones *Sisplay* and *fdisplay*), which formats the output. To add a function one writes the function code and:

- Puts the name in function *reserved*
- Translates the function usage to function call in *evaluate*
- Formats the output as needed in *Display* (and other clones)

Improvements: The symbolic manipulator is very efficient in terms of memory and minimization. But still, in future this might not be adequate. There is scope for further improvement in these areas.

1) Memory: In the design it was specified that (except for *inverse*) the symbolic manipulator frees the *Table* only between two declarations. It is possible to free some portion of the *Table* even inside a single declaration. For this purpose we can either reuse or free the *Tokens* which are no longer a part of the expression. This has to be done carefully because even if a *Token* is currently not a part of the expression tree, it may be later. In such cases one can *lock* those *Tokens* by making the *valid* field, say, 1 and releasing the other unused *Tokens*.

2) Minimization: The present implementation recognizes only 2^n of n! combinations as identical in case of commutative operators. To recognize all n! combinations, one can sort the expressions on the key provided in the *key* field of the *listnode* and then compare the expressions. For the cases of factors, it is possible to de-factorize the expressions, sort them and compare. These two improvements are sure to increase its memory efficiency too, as the two are interrelated. In addition there is hardly any bound on the trigonometric and other simplifications possible.

Bibliography

[1] Faux, I. D., and Pratt, M. J.,1983, *Computational Geometry for Design and Manufacture*, Ellis Horwood Ltd., Chichester.

[2] Aho, A. V., Hopcroft, J. E. and Ullman, J. D., *The Design and analysis of computer algorithms.*, Addison-Wesley, 1974.

[3] Forrest, A. R. 1971, *Computational Geometry ,Proceedings of Royal Society, London, A321, pp. 187 - 195.*

[4] Mantyla, M. 1988, *An Introduction to Solid Modeling, Computer Science Press, Inc.*

[5] Mortenson, M. E., 1985, *Geometric Modeling, John Wiley and Sons, New York.*

[6] Wozny, M. J., Turner, J. U. and Preiss, K., (editors), 1990, *Geometric Modeling for Product Engineering,* North-Holland.

[7] Boltyanskii, V, C., 1964,*Envelopes*, Macmillan Company.

[8] Chakraborthy, J., and Dhande, S. G., 1977, *Kinematics and Geometry of Planar and Spatial Cam Mechanisms*, Wiley Eastern Limited, New Delhi.

[9] Misra, B. K., Pradhan, B. S. S. and Dhande, S. G., 1992, *Computational Conjugate Geometry of Flexible Generating curves, Proceedings of* 5^{th} *ASEE International Conference on Engineering Computer Graphics and Descriptive Geometry, August 17-21, RMTI, Melbourne, Australia, pp. 428-432.*

[10] Shinan, L. and Jinguange, Z., 1990, *Envelope Method on the CAD of Profile Cutters, Proceedings of the ASEE conference on Computer Graphics*A New Vision on Engineering, Miami, FL, pp. 398-404.

[11] Voruganti, R. S., 1990, *Symbolic and Computational Conjugate Geometry for Design and Manufacturing Applications, M.S.Thesis, Virginia Polytechnic Institute and State University.*

[12] Ashrafiuon, H. and Mani N. K., 1989, *Applications of Symbolic Computing for Numerical Analysis of Mechanical Systems, Applied Mechanisms & Robotics conference, Cincinnati,pp. 141-149.*

[13] Brackx, F. and Constales, D., 1991, *Computer Algebra with Lisp and Reduce: An Introduction to Computer Aided Mathemetics,* Kluwer Academic Publishers.

[14] Buchberger, B., Collins, G. E. and Loos, R. (editors), 1982, *Computer Algebra - Symbolic and Algebraic Computation,*Springer-Verlag, New York.

[15] Davia V. Chudnovsky and Richard D. Jenks(editors) ,1989, *Computer Algebra : Lecture Notes in Pure and Applied Mathematics - Vol. 113, Marcel Dekker,* New York.

[16] Inada, N. and Soma, T(editors), 1985, *RIKEN's International symposium on Symbolic and Algebraic Computation by Computers,* World Scientific, Philedelphia.

[17] Moses J., 1971b, *Algebraic Simplification : A Guide for the Perplexed , Communications of the ACM , Vol. 18, No.8, PP.527-537.*

[18] Raphael, B., Bobrow, D. G., Fein, L. and Young, J. W., 1966, *A Breif Survey of the Languages for Symbolic and Algebraic Manipulation, Symbol Manipulation Languages and Techniques, Proceedings of the IFIP Working Conference on Symbol manipulation Languages,* North Holland Publishing Company, Amsterdam, pp.1-54.

Appendix A

Formal Specification of Symbolic Manipulator

The following is the formal specification of the symbolic manipulator.

```
%%

%token  IDENTIFIER
%token  CONSTANT
%token  EQ_OP NE_OP
%token   SIN COS TAN COT SEC COSEC ASIN ACOS ATAN
%token   ACOT ASEC ACOSEC SINH COSH
%token   TANH COTH SECH COSECH ASINH ACOSH ATANH
%token   ACOTH ASECH ACOSECH
%token  SQRT MATRIX LOG ABS

%start bgn
%%

primary_expr          : identifier
                      | CONSTANT
```

```
                    | '(' expr ')'
                    | '_' '_' function_definition
                    ;

expo_expr           : primary_expr
                    | primary_expr expo_operator expo_expr
                    ;

expo_operator       : '^'
                    ;

unary_expr          : unary_operator cast_expr
                    ;

unary_operator      : SIN
                    | COS
                    | TAN
                    | COT
                    | SEC
                    | COSEC
                    | ASIN
                    | ACOS
                    | ATAN
                    | ACOT
                    | ASEC
                    | ACOSEC
                    | SINH
                    | COSH
                    | TANH
                    | COTH
```

```
                    | SECH
                    | COSECH
                    | ASINH
                    | ACOSH
                    | ATANH
                    | ACOTH
                    | ASECH
                    | ACOSECH
                    | LOG
                    | SQRT
                    | ABS
                    | '+'
                    | '-'
                    ;

cast_expr           : expo_expr
                    | unary_expr
                    ;

multiplicative_expr : cast_expr
                    | multiplicative_expr '*' cast_expr
                    | multiplicative_expr '/' cast_expr
                    | multiplicative_expr '%' cast_expr
                    ;


additive_expr       : multiplicative_expr
                    | additive_expr '+' multiplicative_expr
                    | additive_expr '-' multiplicative_expr
                    ;
```

```
equality_expr       : additive_expr
                    | equality_expr EQ_OP additive_expr
                    | equality_expr NE_OP additive_expr
                    ;

assignment_expr     : equality_expr
                    | unary_expr assignment_operator assignment_expr
                    ;

assignment_operator : '='
                    ;

expr                : assignment_expr
                    ;

declaration         : my_type_spec mat_init_declarator ';'
                    | init_declarator ';'
                    ;


init_declarator     : declarator
                    | declarator '=' initializer
                    ;

mat_init_declarator : mat_declarator
                    | mat_declarator '=' mat_initializer
                    ;
```

```
declarator          : identifier
                    | declarator '(' identifier_list ')'
                    | declarator '('  ')'
                    ;

mat_declarator      : identifier '[' my_const_list ']'
                    ;

my_const_list       : CONSTANT
                    | my_const_list ',' CONSTANT
                    ;

identifier_list     : identifier
                    | identifier_list ',' identifier
                    ;

mat_initializer     : expr
                    | '{' mat_initializer_list '}'
                    ;

mat_initializer_list: mat_initializer
                    | mat_initializer_list ',' mat_initializer
                    ;

initializer         : expr
                    | '{' initializer '}'
                    ;

compound_statement  : '{' expression_statement '}'
                    ;
```

```
expression_statement: ';'
                    | expr ';'
                    ;

bgn             : file
                    ;

file                : external_definition
                    | file external_definition
                    ;

external_definition : function_definition
                    | declaration
                    ;

function_definition : declarator function_body
                    ;

my_type_spec        : MATRIX
                    ;

function_body       : compound_statement
                    ;

identifier          : IDENTIFIER
                    ;
%%
```

Appendix B

A Sample Input to Symbolic Manipulator

Here we give a sample input of the symbolic manipulator.

```
PI = 3.14159265;
DEG= PI/180.0;
PZERO = 1.0E-10;
INF = 1.0E+20;
TRUE = 1.0;
FALSE = 0.0;

/*CUTTER PARAMETERS*/

IFR = TRUE;
d = 1.0;
r = PZERO;
a=PZERO;
b=0.0;
h1=70.0;
einit =0.5;
finit = 0.0;
```

```
hinit=0.0;


/*KINEMATICS*/

theta1 = -PI/2.0;
phi1 = 0.0;
xi1 = PI/varparam8;

theta2 = t * varparam1 ;
phi2 = 0.0;
xi2 = 0.0;

a1=0.0;
b1= varparam3 * (1.0 + varparam5*t + cos(varparam4*t*PI/180.0));
c1=0.0;

a2= -varparam2 * t * varparam1/(2.0 * PI);
b2=0.0;
c2=0.0;


boolb = (abs(b)/(abs(b) + 1.0E-20))^0.00001;
notb = abs(boolb - 1.0);

e=einit * abs(IFR - 1.0) + IFR*(d/2.0 - r*((cos(a) -
sin(b))/cos(a + b)));

f=finit * abs(IFR - 1.0) + IFR*(r + e*sin(a))/cos(a) ;
```

```
xPc=
(e + f*tan(a))*cos(a)*cos(a) - sqrt(abs(((e + f*tan(a))*cos(a)
*cos(a))^2.0 - (e*e + f*f - r*r)*cos(a)*cos(a)));

zPc=xPc*tan(a);


zQc =
(f + (e + (d/2.0)*(tan(a)*tan(b) - 1.0))*tan(b))*cos(b)*cos(b)
+ sqrt(abs(((f + (e + (d/2.0)*(tan(a)*tan(b) - 1.0))*tan(b))*cos(b)
*cos(b))^2.0 - (f*f + (e + (d/2.0)*(tan(a)*tan(b) - 1.0))^2.0 - r*r)
*cos(b)*cos(b)));

xQc=zQc*tan(b) - (d/2.0)*(tan(a)*tan(b) - 1.0);

h = IFR*notb*zQc + abs(IFR*notb - 1.0)*hinit;


xSc=h*tan(b) - (d/2.0)*tan(a)*tan(b) + (d/2.0);

zSc=h;


s1=sqrt(abs(xPc*xPc + zPc*zPc));

matrix tantemp1[1.0,2.0]={zQc - f , xQc -e};

matrix tantemp2[1.0,2.0]={zPc - f , xPc -e};
```

```
s2=abs(r*(__taninv(){tantemp1;} - __taninv(){tantemp2;}));

s3=sqrt(abs((xSc - xQc)^2.0 + (zSc - zQc)^2.0));

s4=abs(h1);


matrix cPu1[1.0,4.0]={xc1,yc1,zc1,1.0};

matrix cPu2[1.0,4.0]={xc2,yc2,zc2,1.0};

matrix cPu3[1.0,4.0]={xc3,yc3,zc3,1.0};

matrix cPu4[1.0,4.0]={xc4,yc4,zc4,1.0};


xc1=u*cos(a);

yc1=0.0;

zc1=u*sin(a);


matrix tantemp3[1.0,2.0]={s1*sin(a) - f , s1*cos(a) - e};

xc2=e + abs(r)*cos(__taninv(){tantemp3;} + ((u - s1)/r));

yc2=0.0;

zc2=f + abs(r)*sin(__taninv(){tantemp3;} + ((u - s1)/r));
```

```
xc3=(u - (s1 + s2 + s3))*sin(b) + xSc ;

yc3=0.0;

zc3=(u - (s1 + s2 + s3))*cos(b) + zSc ;


xc4=xSc;

yc4=0.0;

zc4=u - (s1 + s2 + s3) + zSc ;

matrix M1c[4.0,4.0]=
{cos(v),sin(v),0.0,0.0,-sin(v),cos(v),0.0,
0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0};

p11= __mmult(){cPu1*M1c ;};

p12= __mmult(){cPu2*M1c ;};

p13= __mmult(){cPu3*M1c ;};

p14= __mmult(){cPu4*M1c ;};

matrix p11[1.0,4.0]={u*cos(v),u*sin(v),a*u*u + b*u + c,1.0};

matrix p12[1.0,4.0]={u*cos(v),u*sin(v),a*u*u + b*u + c,1.0};
```

```
matrix p13[1.0,4.0]={u*cos(v),u*sin(v),a*u*u + b*u + c,1.0};

matrix p14[1.0,4.0]={u*cos(v),u*sin(v),a*u*u + b*u + c,1.0};

itheta2 =  -theta2 ;

iphi2 = -phi2;

ixi2 = -xi2;

ia2= -a2;

ib2= -b2;

ic2= -c2;

matrix R01x[4.0,4.0]=
{1.0,0.0,0.0,0.0,0.0,cos(theta1),sin(theta1)
,0.0,0.0,-sin(theta1),cos(theta1),0.0,0.0,0.0,0.0,1.0};

matrix R01y[4.0,4.0]=
{cos(phi1),0.0,-sin(phi1),0.0,0.0,1.0,0.0,
0.0,sin(phi1),0.0,cos(phi1),0.0,0.0,0.0,0.0,1.0};

matrix R01z[4.0,4.0]=
{cos(xi1),sin(xi1),0.0,0.0,-sin(xi1),cos(xi1)
,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0};

matrix T01[4.0,4.0]=
```

```
{1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,
0.0,a1,b1,c1,1.0};

M01 = __mmult(){R01x*__mmult(){R01y*__mmult(){R01z*T01;};};};

M01d = __differentiate(t){M01;};

matrix R02x[4.0,4.0]=
{1.0,0.0,0.0,0.0,0.0,cos(theta2),sin(theta2),
0.0,0.0,-sin(theta2),cos(theta2),0.0,0.0,0.0,0.0,1.0};

matrix R02y[4.0,4.0]=
{cos(phi2),0.0,-sin(phi2),0.0,0.0,1.0,0.0,0.0,
sin(phi2),0.0,cos(phi2),0.0,0.0,0.0,0.0,1.0};

matrix R02z[4.0,4.0]=
{cos(xi2),sin(xi2),0.0,0.0,-sin(xi2),cos(xi2),
0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0};

matrix T02[4.0,4.0]=
{1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,
0.0,a2,b2,c2,1.0};

M02 = __mmult(){R02x*__mmult(){R02y*__mmult(){R02z*T02;};};};

M02d = __differentiate(t){M02;};

matrix iR02x[4.0,4.0]=
{1.0,0.0,0.0,0.0,0.0,cos(itheta2),sin(itheta2)
,0.0,0.0,-sin(itheta2),cos(itheta2),0.0,0.0,0.0,0.0,1.0};
```

```
matrix iR02y[4.0,4.0]=
{cos(iphi2),0.0,-sin(iphi2),0.0,0.0,1.0,0.0,
0.0,sin(iphi2),0.0,cos(iphi2),0.0,0.0,0.0,0.0,1.0};

matrix iR02z[4.0,4.0]=
{cos(ixi2),sin(ixi2),0.0,0.0,-sin(ixi2),cos(ixi2)
,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0};

matrix iT02[4.0,4.0]=
{1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,
ia2,ib2,ic2,1.0};

M20 = __inverse(){M02;};

/*M20 = __mmult(){iT02*__mmult(){iR02z*__mmult(){iR02y*iR02x;};};};
 would be more efficient in terms of memory*/

p21tmp= __mmult(){M01*M20 ;};

p21 = __mmult(){p11*p21tmp;};

p22 = __mmult(){p12*p21tmp;};

p23 = __mmult(){p13*p21tmp;};

p24 = __mmult(){p14*p21tmp;};


N1p11 = __cross(){__differentiate(u){p11;}*__differentiate(v){p11;};};
```

```
N1p12 = __cross(){__differentiate(u){p12;}*__differentiate(v){p12;};};

N1p13 = __cross(){__differentiate(u){p13;}*__differentiate(v){p13;};};

N1p14 = __cross(){__differentiate(u){p14;}*__differentiate(v){p14;};};

N0p11=__mmult(){N1p11*M01;};

N0p12=__mmult(){N1p12*M01;};

N0p13=__mmult(){N1p13*M01;};

N0p14=__mmult(){N1p14*M01;};

N2p11=__mmult(){N0p11*M20;};

N2p12=__mmult(){N0p12*M20;};

N2p13=__mmult(){N0p13*M20;};

N2p14=__mmult(){N0p14*M20;};

V11tmpm = __mmult(){p11*M01d;};

V21tmpm = __mmult(){p21*M02d;};

V0p121 =__msub(){V21tmpm - V11tmpm;};

V12tmpm = __mmult(){p12*M01d;};
```

```
V22tmpm = __mmult(){p22*M02d;};

V0p122 =__msub(){V22tmpm - V12tmpm;};

V13tmpm = __mmult(){p13*M01d;};

V23tmpm = __mmult(){p23*M02d;};

V0p123 =__msub(){V23tmpm - V13tmpm;};

V14tmpm = __mmult(){p14*M01d;};

V24tmpm = __mmult(){p24*M02d;};

V0p124 =__msub(){V24tmpm - V14tmpm;};

coc1= __dot(){N0p11*V0p121;};

coc2= __dot(){N0p12*V0p122;};

coc3= __dot(){N0p13*V0p123;};

coc4= __dot(){N0p14*V0p124;};
```